**Mario:** Olá pessoas! Sou Mario Souto, e boas-vindas a mais uma Imersão React aqui da Alura!

Dessa vez, faremos um projeto mais sensacional que na última edição! E não estou sozinho para fazer isso, afinal vocês sabem que sempre há mais alguém na Imersão, e a pessoa especial que está conosco é o Paulo Silveira!

**Paulo:** Olá Mario! Sou o Paulo Silveira, CEO da Alura e estarei colocando perguntas para o Mario nessa Imersão junto com você que está estudando. Até nos separamos em dois estúdios diferentes.

E estou muito contente, pois quem para quer dar um passo a mais em frameworks de front-end, é uma ótima oportunidade de trabalho! Pois é como as pessoas costumam entrar no primeiro e segundo emprego nessa área de tecnologia, pegando CSS, JavaScript e HTML.

O React, sem dúvida, é uma das maiores bibliotecas, se não a maior de todas. Hoje em dia temos outras escutas do View e do Angular, inclusive o Next que veremos junto com o React. Mas com certeza está no mercado inteiro.

Há muitas vagas de trabalho, e não é à toa que temos muitos depoimentos de alunos e alunas que passaram pela Imersão React ou pelas formações de front-end da Alura, e conseguiram seu objetivo.

Claro que não é uma promessa, afinal ninguém dominará o React perfeitamente em apenas cinco dias, mas há muito conteúdo importante. Você criará realmente um projeto completo.

Nossa intenção é que você saia dessa Imersão com um bom projeto para o seu portfólio no ar do GitHub ou do Cloud, e assim poderá mostrar para as pessoas, desde quem for te contratar até seus amigos e amigas: "olha o que eu fiz, olha o que eu aprendi, olha de onde eu comecei até onde eu cheguei".

Afinal, as empresas hoje buscam a sua capacidade de aprender, e não necessariamente o quanto você já conhece da biblioteca. Claro que também é importante e aprenderemos sobre isso nesta Imersão, porém é mais importante ainda mostrar a sua habilidade em ir além e de querer conhecer o que está por trás dos fundamentos do React.

**Mario:** Focamos tanto em aprender os nomes certos, quanto o porquê de usar ou não as ferramentas. Se você conseguir dizer essas coisas em entrevistas, passará um sinal de conhecimento muito maior para o entrevistador do que só adicionar várias coisas no currículo.

Porém, no dia-a-dia de trabalho, é a sua capacidade real que irá resolver o problema de fato, e não necessariamente a quantidade de nomes que você sabe. Este é um ponto importante de frisar, e que cada vez mais é cobrado nas entrevistas.

**Paulo:** Exatamente, e iremos passar por isso. Você verá que é muito importante aprender assim. Esta é a metodologia da Alura, em que trazemos o Dev em T como mostraremos adiante para você compreender o que está acontecendo por "debaixo dos panos".

Claro que em começo de carreira isso acontece aos poucos, mas não queremos que você apenas use os frameworks, e sim que vá além e entenda tudo o que está acontecendo, seja no navegador, V8, TypeScript e etc.

É bastante conteúdo, mas avançaremos com calma para você realmente já colocar no ar o começo do nosso projeto feito nesta aula, e a partir disso iremos aprimorar e entender melhor alguns recursos, como o React funciona, o que você precisa entender do JavaScript, do ciclo de vida e assim por diante.

É fundamental ter domínio, e é sempre muito bacana quando conhecemos pessoas que são muito boas em suas profissões, mas é ainda mais legal quando de fato dominam o funcionamento da ferramenta.

**Mario:** Perfeito, ótima introdução. Vamos tentar dar os primeiros passos passando um pouco da nossa visão, como pensamos, como fazemos, como procuramos, como lemos a documentação e etc.

Todos esses detalhes irão somar na sua evolução das suas habilidades.

**Paulo:** E Mario, antes de mostrarmos o site do React, o que é o GitHub e passarmos o projeto, traremos dois depoimentos muito importantes de pessoas que estudaram conosco na Imersão e nos cursos de front-end da Alura.

Assim você entenderá de onde podemos sair e até onde podemos chegar. É claro, contando muito com o seu esforço e dedicação. Então vamos lá!

Ana Paula: A minha experiência da Imersão foi muito positiva! Há um ano atrás eu comecei a faculdade e comecei a estudar um pouco, e quando chegou na parte de programação eu me assustei um pouco.

Talvez a didática usada pelo professor não tenha sido a melhor, e tive um bloqueio pensando que não era para mim. Mas resolvi dar uma segunda chance e me surpreendi!

Gabriel: Antes da Alura eu estava um pouco perdido no meio tech, e não sabia para onde ir. Eu queria muito estudar React, já que estava como autodidata aprendendo sobre HTML e CSS, então assinei a Alura para aprofundar mais, além de Python e questões de acessibilidade.

Intensifiquei o estudo em React, o que me ajudou muito no mercado de trabalho. Até consegui minha primeira vaga na área de tecnologia. Acredito que, sem a Alura, tudo seria muito mais difícil.

Inclusive, em entrevistas anteriores, alguns entrevistadores comentaram sobre eu ter estudado na Alura, e perguntavam sobre isso. Às vezes as empresas fornecem o acesso à plataforma para os funcionários, e isso confirma a credibilidade da escola no mercado.

Então realmente foi um grande diferencial para mim.

**Paulo:** Legal né, Mario? Queremos trazer essa mesma motivação para você, conhecendo o que as alunas e alunos da Alura fazem e fizeram. Claro que as pessoas estão em estágios e têm disponibilidade diferentes, mas o que esperamos é compromisso.

Não à toa estamos fazendo esse grande evento, pois recomendamos que você se dedique e reserve pelo menos duas horas ao dia para assistir às aulas, praticar ao máximo, colocar seu projeto no ar, conversar conosco no Discord e publicar seu desenvolvimento.

Nos marque no LinkedIn e no Instagram, e também lembre-se de registrar sua evolução, pois isso realmente mostrará seu portfólio e servirá como uma "vitrine" do seu trabalho, como mostraremos com projetos de outras pessoas ao longo do curso. Então vamos lá!

**Mario:** A Alura está com tema do filme Matrix, o que tem a ver com o projeto que faremos nesta Imersão, mas o seu pode ser diferente se quiser.

Começaremos mostrando a documentação do React, pois muitos dizem que se deve começar pela documentação quando iniciamos o trabalho com uma tecnologia.

Como o Paulo sempre fala, é interessante termos uma noção da ferramenta antes de começarmos a escrever o código. Nem sempre precisa ler a documentação inteira, e basta clicar em "Comece a Usar" no site do React para já começar a ter uma ideia de como funciona.

Conforme a necessidade, vamos nos aprofundar nos recursos, afinal ler tudo sem praticar não é suficiente para absorver o conteúdo. É sempre importante unir estudo e prática para montar um bom aprendizado.

Paulo, você que está fazendo as perguntas, O React Costuma causar algumas dúvidas sobre o que é exatamente, se é JavaScript ou HTML.

**Paulo:** Exato, o que me causa surpresas no React e em outros frameworks, é que o navegador não entende esse código. Temos uma classe com um retorno sem aspas que abre chaves e inicializa uma tag, e isso não é um código JavaScript compatível. Até o nome deste arquivo se chama JSX.

**Mario:** JSX se traduz como "JavaScript XML", o que é interessante saber.

**Paulo:** Os arquivos que iremos escrever com este código React não será entendido pelo navegador, e de alguma forma, algo acontecerá para gerar HTML, CSS e JavaScript de forma que o Chrome por exemplo possa entender, conforme aprendemos nas primeiras aulas de front-end.

**Mario:** E quem faz essa transformação é uma ferramenta chamada Babel. Então se estivermos trabalhando com algo de React e nos depararmos com babel-config seguido de outra coisa, é por conta desse compilador.

Antigamente chamávamos de transpilador, pois traduzia essa sintaxe que não existe para algo que existe, e atualmente chamamos de compilador. Há toda uma estrutura próxima, e há outras linguagens que também utilizam o Babel.

Porém, no caso do React, há o plugin babel-react que é capaz de traduzir todos esses códigos para as chamadas de função que o navegador entende e pode processar.

**Paulo:** Ainda não precisamos muito saber disso, afinal o Babel nem vai aparecer nesta aula. Inclusive, hoje em dia muitas pessoas aprendem a programar direto no react e não sabem que o navegador não lê este código.

Para início de carreira em que já sabe programar um pouco e não entende de JavaScript mas está começando pelo React, não tem problemas.

Porém, daqui a um tempo, será necessário entender melhor como este código vira algo que o navegador entende. Certo?

**Mario:** Outro ponto importante é que - e é uma experiência pessoal - quando eu comecei a aprender React, foquei muito em tentar aprender todo o ferramental, que na época era bem iniciante. Gastei cerca de dois meses tentando entender tudo, e hoje o ferramental atualizou e o React não.

Ou seja, demorei meses para aprender algo que, quando mudou, eu não conseguia entender direito. Então se focarmos na ferramenta que nos ajudará a trabalhar e entregar resultados que é o React de fato, essa parte que está por "debaixo dos panos" vão ser compreendidas aos poucos.

Invariavelmente teremos que resolver problemas no dia-a-dia, como otimizar uma imagem para fazer algo por exemplo, então precisaremos utilizar melhor.

Com isso, resolvemos as questões sob demanda. É bom evitar pegar tudo de uma vez, pois pode ser que às vezes não haja um bom motivo para se dedicar àquilo.

Então, encontrar a motivação do porquê estamos estudando é fundamental. Portanto, primeiro aprendemos sobre a ferramenta, depois teremos que resolver problemas ou simplesmente nos aprofundar, e seguimos descendo as "camadas" para trabalhar.

Lembrando que a curiosidade é muito importante, além de um grande diferencial. Mas outro ponto interessante é que, como foi no meu caso em que tive de aprender a fazer por conta de antigamente ser bem mais "manual", não havia uma forma clara dos passos para um projeto React ou Angular.

Mostraremos uma maneira atual de iniciar um projeto usando o Next.js. Inclusive, no site encontramos o título em inglês dizendo que é o "framework de React pronto para produção".

O Next.js se posiciona desta forma porque tem uma "casca" para trabalharmos que tem vários recursos que nos ajudam tanto no desenvolvimento quanto na hora da compilação da aplicação para produção.

Ou seja, organiza o Babel e usa várias outras ferramentas de otimização que auxiliam muito, e se dizem também como "SDK da web", o "kit" de desenvolvimento web.

Portanto, se quisermos fazer um site, irão querer que usemos o Next.js e, diferente de muitos debates sobre qual ferramenta serve para qual tamanho de projeto, o Next.js tem um conjunto de funcionalidades que auxiliam na construção de acordo com a escala.

No básico, será apenas uma ferramenta para trabalhar com React, e conforme o projeto aumenta como novas páginas ou chamadas ao longo do tempo e a necessidade cresce, possui suporte nativo que nos permite explorar a documentação para resolver as questões e adicionar recursos.

**Paulo:** Só para entender, quando formos fazer a instalação para criar o projeto, teremos o Next.js na linha de comando. Porém é um framework "em cima" da biblioteca do React.

Ou seja, tudo o que uma pessoa sabe sobre React pode ser usado em nosso projeto da Imersão.

**Mario:** Exatamente, todo conhecimento que houver sobre React funcionará no Next.js, pois esta é a proposta da "camada externa" no código para ajudar a fazer algumas coisas, principalmente de building timing, lidar com páginas, roteamento e outras coisas que, no passado, aparecia muito ao configurar, e hoje em dia há outras soluções mais "opinionated", ou seja, "opinativas", em que alguém define como deve ser e nós só seguimos e trabalhamos.

**Paulo:** Mas por que usar o React e não usar o javascript normal, além dos mecanismos clássicos da internet? É justamente porque facilita o trabalho, pois o HTML, CSS e JavaScript ficaram muito complexos com o tempo.

Inclusive o criador do React é o Facebook, que é uma rede social com uma enorme quantidade de funcionalidades em que cada componente visual possui alguma reação de acordo com algo que acontece, seja no server side, no back-end, ou no visual do mouse over. Então começou a ficar bem complicado gerenciar tudo em um gigantesco HTML e CSS.

**Mario:** É exatamente isso. Pensando nestes "pedaços" de tela, o mais relevante não somente no React, mas também em todas as ferramentas atuais, essa ideia de componentes.

Falo como ideia, pois pode ser que aprendamos React hoje e entremos em um projeto que use View e Angular por exemplo, afinal as empresas têm diferentes tecnologias. Portanto sempre focaremos em ideias nas Imersões.

É interessante focar nos componentes porque há várias bibliotecas open source que nos permitem pegar os trechos de código e aplicar nos projetos. Claro que algumas exigem algumas configurações seguindo a documentação, mas por exemplo o Material UI, possui vários componentes baseados no material design do Google e Android, e acessando sua documentação, encontramos a parte de "Componentes".

**Paulo:** A página inicial do Material UI mostra os componentes de calendário, temperatura, botão de "comprar" e outros.

**Mario:** E ainda é interativo! Há imagens e botões para clicarmos.

**Paulo:** E com cada tag, o React sabe pegar e gerar JavaScript e etc. para criar esse componente visual, ao invés de copiarmos e colarmos de vários lugares, ou escrevermos nosso próprio.

Não necessariamente os componentes são visuais, mas na maioria dos casos é. Então cada um estará pronto para trabalharmos no React.

**Mario:** E são infinitos! Há até bibliotecas que nos ajudam a catalogar esses componentes. Usaremos um conjunto no Storyboard ao longo do projeto, e podemos clicar e mostrar o código para podermos testar e definir o que queremos.

Então há muitos recursos focados atualmente em ajudar na documentação, pois muitos times possuem apenas um pequeno botão, mas quando vamos para empresas maiores, pode ser que precisemos fazer uma página nova de produtos, por exemplo.

A própria equipe de design já criou e selecionou alguns templates. Depois, mandam para a equipe de desenvolvimento juntá-los para criar a página. Desta forma, teremos mais autonomia dentro do ecossistema do React, tanto na documentação quanto nos componentes que nos ajuda a reutilizar código e escrever em cima de algo que já está pronto, como o calendário do Airbnb que é usado em várias companhias aéreas.

É muito interessante conseguir começar a enxergar e identificar a origem dos componentes. Então realmente o open source deu bastante certo e se expande cada vez mais, com mais recursos e organização.

Teremos ícones e várias outras coisas, mas nosso projeto especial ainda não foi mostrado. Com o link, poderemos ver a página com os componentes que poderão ser customizados. É muito inspirado na página da Alura.

Basicamente, pegamos a ideia de trabalhar tanto com os componentes quanto com as bases do React,e a questão de reatividade e algo mudar outra coisa, resolvemos fazer um chat inspirado no Discord, mas que você dará o tema.

O nosso foi inspirado no Alura Matrix e nas ideias do Paulo, e possui todo o processo de digitarmos o nome de usuário do GitHub. Usaremos a app do Matrix que terá um lugar que buscará os dados das mensagens, as fotos do GitHub e etc.

Fizemos uma grande interação que será construída com vocês. Quando clicarmos em "Entrar", exibiremos a parte interna do site com as mensagens, botão de logout, campo de texto e etc. Tudo isso iremos construir juntos ao longo das aulas da Imersão.

**Mario:** Olá pessoas! Sou Mario Souto, e boas-vindas a mais uma Imersão React aqui da Alura!

Dessa vez, faremos um projeto mais sensacional que na última edição! E não estou sozinho para fazer isso, afinal vocês sabem que sempre há mais alguém na Imersão, e a pessoa especial que está conosco é o Paulo Silveira!

**Paulo:** Olá Mario! Sou o Paulo Silveira, CEO da Alura e estarei colocando perguntas para o Mario nessa Imersão junto com você que está estudando. Até nos separamos em dois estúdios diferentes.

E estou muito contente, pois quem para quer dar um passo a mais em frameworks de front-end, é uma ótima oportunidade de trabalho! Pois é como as pessoas costumam entrar no primeiro e segundo emprego nessa área de tecnologia, pegando CSS, JavaScript e HTML.

O React, sem dúvida, é uma das maiores bibliotecas, se não a maior de todas. Hoje em dia temos outras escutas do View e do Angular, inclusive o Next que veremos junto com o React. Mas com certeza está no mercado inteiro.

Há muitas vagas de trabalho, e não é à toa que temos muitos depoimentos de alunos e alunas que passaram pela Imersão React ou pelas formações de front-end da Alura, e conseguiram seu objetivo.

Claro que não é uma promessa, afinal ninguém dominará o React perfeitamente em apenas cinco dias, mas há muito conteúdo importante. Você criará realmente um projeto completo.

Nossa intenção é que você saia dessa Imersão com um bom projeto para o seu portfólio no ar do GitHub ou do Cloud, e assim poderá mostrar para as pessoas, desde quem for te contratar até seus amigos e amigas: "olha o que eu fiz, olha o que eu aprendi, olha de onde eu comecei até onde eu cheguei".

Afinal, as empresas hoje buscam a sua capacidade de aprender, e não necessariamente o quanto você já conhece da biblioteca. Claro que também é importante e aprenderemos sobre isso nesta Imersão, porém é mais importante ainda mostrar a sua habilidade em ir além e de querer conhecer o que está por trás dos fundamentos do React.

**Mario:** Focamos tanto em aprender os nomes certos, quanto o porquê de usar ou não as ferramentas. Se você conseguir dizer essas coisas em entrevistas, passará um sinal de conhecimento muito maior para o entrevistador do que só adicionar várias coisas no currículo.

Porém, no dia-a-dia de trabalho, é a sua capacidade real que irá resolver o problema de fato, e não necessariamente a quantidade de nomes que você sabe. Este é um ponto importante de frisar, e que cada vez mais é cobrado nas entrevistas.

**Paulo:** Exatamente, e iremos passar por isso. Você verá que é muito importante aprender assim. Esta é a metodologia da Alura, em que trazemos o Dev em T como mostraremos adiante para você compreender o que está acontecendo por "debaixo dos panos".

Claro que em começo de carreira isso acontece aos poucos, mas não queremos que você apenas use os frameworks, e sim que vá além e entenda tudo o que está acontecendo, seja no navegador, V8, TypeScript e etc.

É bastante conteúdo, mas avançaremos com calma para você realmente já colocar no ar o começo do nosso projeto feito nesta aula, e a partir disso iremos aprimorar e entender melhor alguns recursos, como o React funciona, o que você precisa entender do JavaScript, do ciclo de vida e assim por diante.

É fundamental ter domínio, e é sempre muito bacana quando conhecemos pessoas que são muito boas em suas profissões, mas é ainda mais legal quando de fato dominam o funcionamento da ferramenta.

**Mario:** Perfeito, ótima introdução. Vamos tentar dar os primeiros passos passando um pouco da nossa visão, como pensamos, como fazemos, como procuramos, como lemos a documentação e etc.

Todos esses detalhes irão somar na sua evolução das suas habilidades.

**Paulo:** E Mario, antes de mostrarmos o site do React, o que é o GitHub e passarmos o projeto, traremos dois depoimentos muito importantes de pessoas que estudaram conosco na Imersão e nos cursos de front-end da Alura.

Assim você entenderá de onde podemos sair e até onde podemos chegar. É claro, contando muito com o seu esforço e dedicação. Então vamos lá!

Ana Paula: A minha experiência da Imersão foi muito positiva! Há um ano atrás eu comecei a faculdade e comecei a estudar um pouco, e quando chegou na parte de programação eu me assustei um pouco.

Talvez a didática usada pelo professor não tenha sido a melhor, e tive um bloqueio pensando que não era para mim. Mas resolvi dar uma segunda chance e me surpreendi!

Gabriel: Antes da Alura eu estava um pouco perdido no meio tech, e não sabia para onde ir. Eu queria muito estudar React, já que estava como autodidata aprendendo sobre HTML e CSS, então assinei a Alura para aprofundar mais, além de Python e questões de acessibilidade.

Intensifiquei o estudo em React, o que me ajudou muito no mercado de trabalho. Até consegui minha primeira vaga na área de tecnologia. Acredito que, sem a Alura, tudo seria muito mais difícil.

Inclusive, em entrevistas anteriores, alguns entrevistadores comentaram sobre eu ter estudado na Alura, e perguntavam sobre isso. Às vezes as empresas fornecem o acesso à plataforma para os funcionários, e isso confirma a credibilidade da escola no mercado.

Então realmente foi um grande diferencial para mim.

**Paulo:** Legal né, Mario? Queremos trazer essa mesma motivação para você, conhecendo o que as alunas e alunos da Alura fazem e fizeram. Claro que as pessoas estão em estágios e têm disponibilidade diferentes, mas o que esperamos é compromisso.

Não à toa estamos fazendo esse grande evento, pois recomendamos que você se dedique e reserve pelo menos duas horas ao dia para assistir às aulas, praticar ao máximo, colocar seu projeto no ar, conversar conosco no Discord e publicar seu desenvolvimento.

Nos marque no LinkedIn e no Instagram, e também lembre-se de registrar sua evolução, pois isso realmente mostrará seu portfólio e servirá como uma "vitrine" do seu trabalho, como mostraremos com projetos de outras pessoas ao longo do curso. Então vamos lá!

**Mario:** A Alura está com tema do filme Matrix, o que tem a ver com o projeto que faremos nesta Imersão, mas o seu pode ser diferente se quiser.

Começaremos mostrando a documentação do React, pois muitos dizem que se deve começar pela documentação quando iniciamos o trabalho com uma tecnologia.

Como o Paulo sempre fala, é interessante termos uma noção da ferramenta antes de começarmos a escrever o código. Nem sempre precisa ler a documentação inteira, e basta clicar em "Comece a Usar" no site do React para já começar a ter uma ideia de como funciona.

Conforme a necessidade, vamos nos aprofundar nos recursos, afinal ler tudo sem praticar não é suficiente para absorver o conteúdo. É sempre importante unir estudo e prática para montar um bom aprendizado.

Paulo, você que está fazendo as perguntas, O React Costuma causar algumas dúvidas sobre o que é exatamente, se é JavaScript ou HTML.

**Paulo:** Exato, o que me causa surpresas no React e em outros frameworks, é que o navegador não entende esse código. Temos uma classe com um retorno sem aspas que abre chaves e inicializa uma tag, e isso não é um código JavaScript compatível. Até o nome deste arquivo se chama JSX.

**Mario:** JSX se traduz como "JavaScript XML", o que é interessante saber.

**Paulo:** Os arquivos que iremos escrever com este código React não será entendido pelo navegador, e de alguma forma, algo acontecerá para gerar HTML, CSS e JavaScript de forma que o Chrome por exemplo possa entender, conforme aprendemos nas primeiras aulas de front-end.

**Mario:** E quem faz essa transformação é uma ferramenta chamada Babel. Então se estivermos trabalhando com algo de React e nos depararmos com babel-config seguido de outra coisa, é por conta desse compilador.

Antigamente chamávamos de transpilador, pois traduzia essa sintaxe que não existe para algo que existe, e atualmente chamamos de compilador. Há toda uma estrutura próxima, e há outras linguagens que também utilizam o Babel.

Porém, no caso do React, há o plugin babel-react que é capaz de traduzir todos esses códigos para as chamadas de função que o navegador entende e pode processar.

**Paulo:** Ainda não precisamos muito saber disso, afinal o Babel nem vai aparecer nesta aula. Inclusive, hoje em dia muitas pessoas aprendem a programar direto no react e não sabem que o navegador não lê este código.

Para início de carreira em que já sabe programar um pouco e não entende de JavaScript mas está começando pelo React, não tem problemas.

Porém, daqui a um tempo, será necessário entender melhor como este código vira algo que o navegador entende. Certo?

**Mario:** Outro ponto importante é que - e é uma experiência pessoal - quando eu comecei a aprender React, foquei muito em tentar aprender todo o ferramental, que na época era bem iniciante. Gastei cerca de dois meses tentando entender tudo, e hoje o ferramental atualizou e o React não.

Ou seja, demorei meses para aprender algo que, quando mudou, eu não conseguia entender direito. Então se focarmos na ferramenta que nos ajudará a trabalhar e entregar resultados que é o React de fato, essa parte que está por "debaixo dos panos" vão ser compreendidas aos poucos.

Invariavelmente teremos que resolver problemas no dia-a-dia, como otimizar uma imagem para fazer algo por exemplo, então precisaremos utilizar melhor.

Com isso, resolvemos as questões sob demanda. É bom evitar pegar tudo de uma vez, pois pode ser que às vezes não haja um bom motivo para se dedicar àquilo.

Então, encontrar a motivação do porquê estamos estudando é fundamental. Portanto, primeiro aprendemos sobre a ferramenta, depois teremos que resolver problemas ou simplesmente nos aprofundar, e seguimos descendo as "camadas" para trabalhar.

Lembrando que a curiosidade é muito importante, além de um grande diferencial. Mas outro ponto interessante é que, como foi no meu caso em que tive de aprender a fazer por conta de antigamente ser bem mais "manual", não havia uma forma clara dos passos para um projeto React ou Angular.

Mostraremos uma maneira atual de iniciar um projeto usando o Next.js. Inclusive, no site encontramos o título em inglês dizendo que é o "framework de React pronto para produção".

O Next.js se posiciona desta forma porque tem uma "casca" para trabalharmos que tem vários recursos que nos ajudam tanto no desenvolvimento quanto na hora da compilação da aplicação para produção.

Ou seja, organiza o Babel e usa várias outras ferramentas de otimização que auxiliam muito, e se dizem também como "SDK da web", o "kit" de desenvolvimento web.

Portanto, se quisermos fazer um site, irão querer que usemos o Next.js e, diferente de muitos debates sobre qual ferramenta serve para qual tamanho de projeto, o Next.js tem um conjunto de funcionalidades que auxiliam na construção de acordo com a escala.

No básico, será apenas uma ferramenta para trabalhar com React, e conforme o projeto aumenta como novas páginas ou chamadas ao longo do tempo e a necessidade cresce, possui suporte nativo que nos permite explorar a documentação para resolver as questões e adicionar recursos.

**Paulo:** Só para entender, quando formos fazer a instalação para criar o projeto, teremos o Next.js na linha de comando. Porém é um framework "em cima" da biblioteca do React.

Ou seja, tudo o que uma pessoa sabe sobre React pode ser usado em nosso projeto da Imersão.

**Mario:** Exatamente, todo conhecimento que houver sobre React funcionará no Next.js, pois esta é a proposta da "camada externa" no código para ajudar a fazer algumas coisas, principalmente de building timing, lidar com páginas, roteamento e outras coisas que, no passado, aparecia muito ao configurar, e hoje em dia há outras soluções mais "opinionated", ou seja, "opinativas", em que alguém define como deve ser e nós só seguimos e trabalhamos.

**Paulo:** Mas por que usar o React e não usar o javascript normal, além dos mecanismos clássicos da internet? É justamente porque facilita o trabalho, pois o HTML, CSS e JavaScript ficaram muito complexos com o tempo.

Inclusive o criador do React é o Facebook, que é uma rede social com uma enorme quantidade de funcionalidades em que cada componente visual possui alguma reação de acordo com algo que acontece, seja no server side, no back-end, ou no visual do mouse over. Então começou a ficar bem complicado gerenciar tudo em um gigantesco HTML e CSS.

**Mario:** É exatamente isso. Pensando nestes "pedaços" de tela, o mais relevante não somente no React, mas também em todas as ferramentas atuais, essa ideia de componentes.

Falo como ideia, pois pode ser que aprendamos React hoje e entremos em um projeto que use View e Angular por exemplo, afinal as empresas têm diferentes tecnologias. Portanto sempre focaremos em ideias nas Imersões.

É interessante focar nos componentes porque há várias bibliotecas open source que nos permitem pegar os trechos de código e aplicar nos projetos. Claro que algumas exigem algumas configurações seguindo a documentação, mas por exemplo o Material UI, possui vários componentes baseados no material design do Google e Android, e acessando sua documentação, encontramos a parte de "Componentes".

**Paulo:** A página inicial do Material UI mostra os componentes de calendário, temperatura, botão de "comprar" e outros.

**Mario:** E ainda é interativo! Há imagens e botões para clicarmos.

**Paulo:** E com cada tag, o React sabe pegar e gerar JavaScript e etc. para criar esse componente visual, ao invés de copiarmos e colarmos de vários lugares, ou escrevermos nosso próprio.

Não necessariamente os componentes são visuais, mas na maioria dos casos é. Então cada um estará pronto para trabalharmos no React.

**Mario:** E são infinitos! Há até bibliotecas que nos ajudam a catalogar esses componentes. Usaremos um conjunto no Storyboard ao longo do projeto, e podemos clicar e mostrar o código para podermos testar e definir o que queremos.

Então há muitos recursos focados atualmente em ajudar na documentação, pois muitos times possuem apenas um pequeno botão, mas quando vamos para empresas maiores, pode ser que precisemos fazer uma página nova de produtos, por exemplo.

A própria equipe de design já criou e selecionou alguns templates. Depois, mandam para a equipe de desenvolvimento juntá-los para criar a página. Desta forma, teremos mais autonomia dentro do ecossistema do React, tanto na documentação quanto nos componentes que nos ajuda a reutilizar código e escrever em cima de algo que já está pronto, como o calendário do Airbnb que é usado em várias companhias aéreas.

É muito interessante conseguir começar a enxergar e identificar a origem dos componentes. Então realmente o open source deu bastante certo e se expande cada vez mais, com mais recursos e organização.

Teremos ícones e várias outras coisas, mas nosso projeto especial ainda não foi mostrado. Com o link, poderemos ver a página com os componentes que poderão ser customizados. É muito inspirado na página da Alura.

Basicamente, pegamos a ideia de trabalhar tanto com os componentes quanto com as bases do React,e a questão de reatividade e algo mudar outra coisa, resolvemos fazer um chat inspirado no Discord, mas que você dará o tema.

O nosso foi inspirado no Alura Matrix e nas ideias do Paulo, e possui todo o processo de digitarmos o nome de usuário do GitHub. Usaremos a app do Matrix que terá um lugar que buscará os dados das mensagens, as fotos do GitHub e etc.

Fizemos uma grande interação que será construída com vocês. Quando clicarmos em "Entrar", exibiremos a parte interna do site com as mensagens, botão de logout, campo de texto e etc. Tudo isso iremos construir juntos ao longo das aulas da Imersão.

**Paulo:** E vai trocar essas mensagens mesmo, porque ficarão guardadas em algum lugar. Mesmo sem fazermos back-end, o Mário vai mostrar soluções que dá para acoplar. Mario, abre a página do Dev em T da Alura para darmos uma olhada. Essa é uma página que quando lançamos marcou muito o mercado no Brasil.

Vale a pena você ler o nosso [manifesto Dev em T](https://www.alura.com.br/dev-em-t), ele mostra como pensamos que você pode organizar sua carreira. Por exemplo, em front-end, pense que você vai se aprofundar na sua carreira de front-end, vai entender não só o React, mas também o JavaScript, o TypeScript, vai ler o código fonte de algumas dessas bibliotecas daqui a alguns anos, vai entender como o navegador funciona, o V8; e também vai pegar um pouco de DevOps, entender o tal do CICD para colocar o build, que vamos fazer aqui, vamos fazer o deploy no Cloud, vai entender um pouco do back-end para consumir microsserviços, vamos fazer isso aqui na Imersão.

Hoje em dia, não basta ser aquela pessoa que faz só uma parte do sistema, porque no squad de uma empresa você precisa conversar com outras pessoas que têm trabalhos um pouco diferentes, mas que estão conectados com o seu. Front-end está conectado com back-end.

Se você não conhecer nem um pouco essa outra área, fica batendo cabeça. Não é à toa que ali na home da Alura temos sete escola que têm muito a ver entre si. Programação com Front-End, Data Science, DevOps, UX & Design, é nessa área que as pessoas utilizam muito o Figma para desenhar, você vão ver que deixaremos para vocês o link do arquivo no Figma que explica como deve ser o layout do nosso projeto Aluracord. os designer fazem no Figma e explicam como deve ser o layout.

A partir daí, essa pessoa que trabalha com UX vai entregar o arquivo no Figma para a pessoa que trabalha com front-end e falar: "front-end, se vira!". Então você precisa entender um pouco do Figma.

Está tudo conectado. Essa Imersão está toda conectada, a Alura está toda conectada e é assim que trabalhamos.

**Mario:** Lembrando que não vai ser do dia para a noite. É um conhecimento que se cria com o tempo. Ter curiosidade para saber disso é um caminho sem volta e agora você vai conhecer isso.

Na Imersão, uma de nossas metas é te dar autonomia, então não vamos criar todo o back-end, mas você terá uma ferramenta que ajuda a fazer essa parte. Você pode levar essa solução para o seu time, usar em outros lugares e ver as possibilidades.

Eu até brinco que todo mundo que trabalha com programação cria seu cinto de utilidades, igual ao Batman, com várias ferramentas e utilidades que guardamos com o tempo e usamos nos mais diversos cenários, seja para fazer coleta de dados, para salvar informação etc. O dia a dia começa a nos cobrar para termos soluções ao nosso alcance. Daremos algumas dessas soluções para você ao longo da Imersão.

Falando em dar soluções para a galera, vamos começar o nosso projeto Aluracord?

Vou acessar a [documentação do Next JS](https://nextjs.org/docs/getting-started), gosto bastante do jeito que eles a organizam. Logo no início tem esse "Getting Started", que é a introdução explicando como fazer para criar um projeto com essa ferramenta.

Vou descer a página, repare que logo de cara ele tem uma parte de setup recomendado, ele fala: "Recomendamos que você crie um projeto usando esse create-next-app...".

Nós recomendamos que você faça esses passos do setup depois do seu primeiro projeto da Imersão, porque esse setup traz vários arquivos que, num primeiro momento, podem te confundir.

Sei que é estranho irmos contra a primeira indicação da documentação. Mas é justamente porque queremos trabalhar em você a ideia de ter um pouco mais de conhecimento do que está acontecendo.

Então, vamos pular para a parte do "Manual Setup" e faremos juntos esse setup manual. Para isso, esperamos que na sua máquina já esteja instalado o Visual Studio Code, que é a ferramenta de editor de código que usaremos, o Node também que é superimportante e junto com o Node vai ter essa ferramenta chamada NPM, que você vai conseguir ver no seu terminal e daqui a pouco te ajudaremos a testar.

**Paulo:** Na página dessa primeira aula da Imersão estão disponíveis os links para instalar o Node e o Visual Studio Code seja no Windows ou Mac.

**Mario:** Precisamos ter essas ferramentas, sem elas não dá para rodar o projeto. Lembrando que tanto o Next quando o React são bibliotecas, são códigos que estão na internet, você pode pesquisar no Google por "github react", vai aparecer o link da documentação que está no GitHub, tem a descrição da ferramenta, os arquivos, então teremos uma ferramenta para nos ajudar a baixar esses pedaços de código da internet, essa ferramenta é esse tal de NPM.

Não só o NPM, uma outra que vocês podem ter em suas máquinas é o Yarn, que faz basicamente a mesma coisa. Tanto o NPM quanto o Yarn são agregadores, eles permitem que você procure essas bibliotecas. Se eu procurar, por exemplo, o React no site do Yarn, ele mostra o React, o read me dele e como instalar usando o Yarn. A mesma coisa acontece no site do NPM.

Vamos voltar para o setup manual da documentação do Next e começar a instalar essas coisas e iniciar nosso projeto. Para isso, vou copiar o comando da documentação:

npm install next react react-dom

# or

yarn add next react react-dom

**Mario:** Vou vir aqui na minha área de trabalho e criar uma pasta nova chamada "aluracord" e vou abrir o Visual Studio Code que já está instalado na minha máquina. Na maioria dos sistemas operacionais basta você arrastar a pasta para dentro do Visual Studio Code e ele já abre seu projeto.

Se, por algum motivo, no seu sistema operacional isso não funcionar, para abrir essa pasta que criamos basta vir em "File > Open Folder".

Vou começar a rodar comandos no terminal para iniciarmos o nosso projeto. Para abrir o terminal devemos ir no menu superior e selecionar "View > Terminal". Para facilitar no dia a dia você pode ver, nesse mesmo caminho, qual é o atalho configurado para abrir o terminal.

Por este ser um projeto JavaScript, e termos essas bibliotecas, precisamos de um arquivo que registra todas as bibliotecas e deixa clara a informação de qual versão React e qual versão das bibliotecas esse projeto tem. Para que todo mundo que vá usar o nosso código consiga ter a mesma versão.

**Paulo:** Muitas linguagens têm tipo um arquivo de build, não é? O arquivo que usamos para pedir que algo seja baixado da internet e jogue algo no servidor de determinada forma. Cada linguagem tem mecanismos para escrever esse arquivo de configuração que tem uma cara de script.

**Mario:** Exato. E aqui, para usar essa ferramenta de script, vou usar o Yarn, que está mais configurado aqui na minha máquina, mas na sua pode ter a NPM, você pode usar a que preferir.

Na descrição eu vou deixar um vídeo mostrando como eu configuro meu ambiente. Se você quiser configurar mais próximo do que eu estou usando aqui, pode fazer isso também.

Antes de colar o comando que copiei da documentação, eu vou inicializar esse arquivo que o Paulo falou. No meu terminal do Visual Studio vou escrever yarn init -y. O -y é porque quando você inicializa um projeto ele faz várias perguntas que no momento não faz sentido responder, só queremos começar.

Após escrever esse comando pressionaremos "Enter". E foi criado o arquivo package.json que é o arquivo que vai gerenciar os pacotes do projeto. Esse package.json tem o nome, a versão, o arquivo principal, informa a licença do projeto… Podemos ver mais dele depois.

Agora veremos esse arquivo ser preenchido. Vou colar no terminal aquele comando que copiei da documentação, yarn add next react react-dom.

**Paulo:** É bom quem está nos assistindo usar o Yarn também, na página da Imersão tem o link de instalação.

**Mario:** Exato. Lembrando que neste caso não existe melhor nem pior, é mais uma questão de preferência para o projeto. Na minha máquina tem o Yarn porque uso muito o React Native - a versão de fazer aplicativos do React - que recomenda o uso do Yarn por conta do Facebook, do ecossistema etc. Só um contexto para você que está assistindo e podermos seguir adiante. É algo que você pode explorar depois das aulas da Imersão. A ideia do React Native é você reaprovitar os conhecimentos que você tem de React em uma outra plataforma. Indico fazer esse teste depois.

Logo após rodar o comando de instalação que fizemos, ele adicionou no package.json o que chamamos de dependências, que são as nossas bibliotecas com suas respectivas versões. Assim teremos o controle de saber qual versão estamos usando, para não quebrar nada e tudo funcionar corretamente.

Feito isso, vamos seguir a documentação, onde ele diz: "Abra agora o package.json e adicione os seguintes scripts".

"scripts": {

"dev": "next dev",

"build": "next build",

"start": "next start",

"lint": "next lint"

}

**Mario:** O que cai na parte de build que o Paulo comentou. Teremos comandos para nos ajudar a rodar o projeto, gerar versão de produção dele para colocar no ar, entre outras coisas. Copiaremos esse trecho de código dos scripts e vamos colar no package.json depois das dependências. Lembrando de, antes de colar esse trecho, colocar uma vírgula depois da chave },. Se você não colocar a vírgula vai aparecer um erro meio estranho, então vamos colocar vírgula para garantir.

Perfeito, já temos os scripts. De volta à documentação, ela explica o que cada script faz. Fiquem tranquilos, pois vamos passar por eles, principalmente o dev, o start e o build.

Basicamente, o próximo passo para vermos algo rodando na prática é criar uma pasta "pages" e dentro dela um arquivo index.js com esse conteúdo para criar uma página home, o começo do nosso projeto, que é um site, estamos usando o React para fazer uma aplicação web.

function HomePage() {

return <div>Welcome to Next.js!</div>

}

export default HomePage

**Mario:** Vou copiar esse código da documentação, voltar para o Visual Studio Code e, no menu lateral esquerdo, criar uma pasta "pages" e dentro dessa pasta criar o arquivo index.js onde vou colar o código que copiei.

Feito isso, vamos rodar no terminal o comando yarn dev, que é um dos comandos de script que colamos no código. Esse dev é um atalho para rodar o next dev que é para rodar o modo de desenvolvimento do Next.

Vamos rodar o comando yarn dev, ir para o navegador e entrar no endereço [http://localhost:3000](http://localhost:3000/). Agora conseguimos ver no navegador o nosso código rodando. Fizemos nosso primeiro output, a primeira saída de alguma coisa.

Agora vamos parar e reanalisar o que fizemos. Nós criamos o package.json, instalamos as bibliotecas, definimos os comandos para rodar o projeto, criamos um arquivo e rodamos no terminal.

Comparado à maioria dos projetos de front-end, principalmente com frameworks, esse é um setup que vai bem direto ao ponto para entendermos. Ele guarda muita coisa "por debaixo dos panos" e é mais tranquilo de replicar em casa, de fazer teste para você entender o que é esse package.json, o que são os scripts, convido você a explorar tudo isso.

Mas agora vamos explorar o arquivo index.js. Nele já temos um primeiro código React rodando, se eu mudar qualquer coisa aqui dentro e salvar, sem recarregar o navegador ele já muda, por exemplo, se eu substituir "Welcome to Next.js!" por "Olá pessoas!" e salvar, ele vai exibir essa mudança no navegador. Então, tem toda uma inteligência para facilitar o desenvolvimento e você, literalmente, só sentar, escrever seu código e ir vendo as alterações sendo aplicadas no seu projeto.

O Next é bem esperto nesse ponto e vai ajudar bastante no que chamamos de developer experience, que é você ter uma experiência legal para trabalhar no seu projeto, refatorar as coisas e assim por diante.

À primeira vista, é importante falar que por mais que pareça uma coisa de outro mundo, temos basicamente uma função JavaScript que retorna esse trecho que parece HTML, return <div>Olá pessoas!</div>.

Tentar deixar claro o nome das coisas ajuda bastante. Isso é um componente React. Já temos o nosso primeiro componente, que foi criado pela própria documentação.

// Componente React

function HomePage() {

// JSX

return <div>Olá pessoas!</div>

}

export default HomePage

Nossa meta é, até o final da aula, ter uma estrutura como este layout que temos pronto, com imagem de fundo, texto de boas-vindas, campo para digitar seu usuário do GitHub, um botão "Entrar"... Vocês poderão customizar e fazer os ajustes que querem em seu projeto.

Para isso, vamos começar pelo título "Boas vindas de volta!" e temos também o subtítulo "Discord - Alura Matrix". Vamos tentar escrever isso no arquivo index.js dentro do nosso JSX.

Podemos copiar esse "Boas vindas de volta!" e colá-lo dentro da div que está sendo retornada. Para ficar mais fácil fazer a quebra de linha e indentar melhor, normalmente o pessoal coloca parênteses em volta de tudo.

Porque se você não coloca o parênteses a fonte do código fica mais escura no VS Code. Então vamos abrir e fechar parênteses e escrever todo o código da tela dentro desse parênteses.

// Componente React

function HomePage() {

// JSX

return (

<div>

Boas vindas de volta!

</div>

)

}

export default HomePage

**Mario:** Vou colocar o "Boas vindas de volta!" dentro de uma tag HTML normalmente, dentro de uma tag <h1>, <h1>Boas vindas de volta!</h1>.

Agora, vamos copiar o subtítulo "Discord - Alura Matrix". Vamos colocá-lo no nosso código dentro de uma tag <h2>, <h2>Discord - Alura Matrix</h2>.

// Componente React

function HomePage() {

// JSX

return (

<div>

<h1>Boas vindas de volta!</h1>

<h2>Discord - Alura Matrix</h2>

</div>

)

}

export default HomePage

**Mario:** Vamos salvar e podemos verificar como ficou a nossa página no navegador. Nossa página foi atualizada. Está longe do layout ainda, mas vamos trabalhando aos poucos e entendendo como trabalhar com as coisas aqui.

O React tem essa característica de misturar muitas coisas, estamos escrevendo o HTML no JavaScript e até o CSS vai ficar aqui dentro também.

Para conseguirmos escrever o CSS dentro do JavaScript e começarmos a estilizar a página, vamos trabalhar com uma técnica chamada "CSS in JS". Se você pesquisar por isso no Google Imagens vai vir algumas coisas genéricas, mas vamos tentar ver juntos o jeito que o próprio Next se propõe a resolver esse problema.

O Next tem uma ferramenta chamada "styled jsx". Só de você estar com o Next rodando no seu projeto, ele tem esse styled-jsx e eles mostram aqui na [página do GitHub acessível neste link](https://github.com/vercel/styled-jsx), um exemplo dele, dentro do que seria um componente:

export default () => (

<div>

<p>only this paragraph will get the style :)</p>

{/\* you can include <Component />s here that include

other <p>s that don't get unexpected styles! \*/}

<style jsx>{`

p {

color: red;

}

`}</style>

</div>

)

É basicamente uma função que tem o parênteses e retorna um trecho de HTML, aqui tem uma tag <style jsx>, mas essa não é uma tag style normal do HTML, essa tem o atributo jsx que mostra que é um style diferente do tradicional e podemos fazer coisas a mais aqui dentro.

Vou copiar esse trecho da tag style que está como exemplo no GitHub para começarmos a estilizar algumas coisas no nosso código.

<style jsx>{`

p {

color: red;

}

`}</style>

**Mario:** Se eu só copiar e colar esse estilo aqui dentro da div do index.js, não vai mudar nada porque não tem uma tag p aqui, mas eu posso substituir o p por h1 e começar a trabalhar na nossa tag de título.

Vou salvar e ver como ficou nossa página no navegador. Ele pintou a fonte do nosso título de vermelho. Antes de mostrar o que tem de diferente nesse estilo, Paulo, quero te fazer uma pergunta. Você chegou a sofrer com CSS, de alguma forma, na sua vida?

**Paulo:** Acho que o complicado do CSS é que ele é meio global, por exemplo, quando você define que quer que todo h1 seja pintado de uma cor, todo h1 será pintado daquela cor - tudo bem que h1 geralmente só temos um, o do título, mas vale como exemplo - aí você precisa começar a definir classes...

**Mario:** Hoje em dia, se você coloca o h1 dentro de uma tag <article> é possível ter múltiplos h1s dentro de uma página.

**Paulo:** Olha só, isso é novidade pra mim. E aí se eu quisesse que cada artigo tivesse um h1 diferente eu teria que tomar cuidado com as classes, com os ids etc. Aí fica aquela briga, vocês escreve uma coisa no HTML e precisa lembrar como está no CSS lá no outro arquivo.

Então parece que a proposta do React e de outros frameworks é tentar dar uma juntada. À primeira vista, isso que você está escrevendo me dói um pouco o olho. Estamos mostrando várias linguagens de programação, misturando JavaScript, HTML e CSS no mesmo trecho de código.

Fica meio embolado o que é o quê. Mas entendo que tem vantagens. Pelo que entendi, estamos falando que a cor desse h1 do componente HomePage, que por enquanto é nosso único componente, tem a fonte vermelha. Então entendo que, ao criar outros componentes, o h1 desses outros componentes não ficarão em vermelho porque ele consegue entender e diferenciar isso.

**Mario:** Exato. Boa parte de usarmos essas coisas mais prontas é ganharmos de bandeja esses recursos. Se, no navegador, eu clicar com o botão direito no texto do título e selecionar "Inspecionar", conseguimos ver que só de ter colocado o trecho de código do <style jsx> são adicionadas várias classes no código da página.

O próprio Next, com a lib do style jsx, está gerenciando como os estilos devem ser aplicados na página e ele cria essa questão do escopo sobre o qual você comentou.

Se eu colocar outros h1 na nossa página, todos esses h1s receberão essa cor vermelha, mas se tiver um outro componente que também tem h1, ele não vai passar. Ou seja, ele quebra esses estilos por componente. Só vai receber o estilo, as tags e o componente que você colocou.

Isso nos ajuda a ter que lidar menos com essa responsabilidade de ter que nomear as coisas. Porque é muito comum em projeto CSS o código crescer e por mais que você tente arquitetar, usando metodologias como BEM (Block Element Modifier) ou DRY (Don't Repeat Yourself) no CSS, com o tempo a galera foi vendo que dar nomes para todas as classes do CSS é difícil de manter no longo prazo.

Tanto que projetos de CSS que não usam ferramenta, por exemplo, o Tailwind CSS, que é um que está bastante popular, [ao ver a documentação do Tailwind acessível neste link](https://tailwindcss.com/docs/utility-first), repare que ele traz um exemplo com classe, mostrando como seria o modelo tradicional.

Nesse exemplo vemos que para fazer um chat precisamos dar um nome com várias classes. E mesmo as coisas que são com CSS, muitas pessoas estão usando uma abordagem de ter pequenas classes que têm coisas específicas. Por exemplo, você define que a margem do eixo x é "auto", aí ele vai gerar um CSS baseado nessas classes.

No React, veremos que tem outra forma de escrever, mas tanto as ferramentas de CSS puro quanto as ferramentas do ecossistema do React, estão tentando evitar que você precise se preocupar com o nome da classe. Então você usa alguma coisa de utilitário, descreve o que você quer e o estilo vai aparecer ali.

**Paulo:** Já estamos dando um passo mais profundo. É óbvio que queremos colocar para você, desde já, que não queremos fazer um tutorial em que é só copiar e colar o nosso código sem entender o que está acontecendo.

Nós queremos, a cada passo, mostrar o que está debaixo e que você tenha essa curiosidade. É assim que se vai longe em uma carreira de tecnologia, não tenha dúvidas. Para o seu futuro, você vai precisar entender: "Porque tem esse class aqui? Eu não escrevi isso no meu código, por que ele fez isso?", "Foi para não dar conflito com um outro possível h1 de outro lugar, ele gerou isso em tempo real, a própria biblioteca gerou isso para nós"...

Então, começar a entender as "entranhas" da biblioteca mesmo. Estamos dando só um aperitivo. Tanto que na formação de Next que lançaremos no final dessa Imersão teremos bastante desse conteúdo. Além de uma visão geral de uso do framework, temos muito de entender e construa você mesmo um pedaço do framework".

Nossa metodologia, o modo como as imersões da Alura e todo o Aluraverso ensina é que você seja uma pessoa que quer entender o que está funcionando atrás da biblioteca, que não está satisfeita em apenas usar.

**Mario:** E nessa de construir coisas, vamos começar a criar nossos componentes agora. Falamos bastante que tem coisas diferentes juntas no código, mas também é mais fácil de separar.

Por exemplo, se quisermos dizer que o título da nossa aplicação vai ter um tamanho, cor e outras coisas específicas, podemos vir aqui na parte de cima do código e só criar outra função, podemos chamar de "Title", se quiser deixar em inglês, ou "Titulo" se quiser deixar em português. Use o nome que achar mais pertinente para o código do projeto.

Adicionaremos um return e já teremos a estrutura de um componente. Então, vamos colar o título h1 dentro dele.

function Title() {

return (

<h1> Boas vindas de volta!</h1>

);

}

**Mario:** O que eu quero fazer aqui não é só criar uma cópia, quero dizer que teremos o nosso jeito de fazer título.

**Paulo:** Só para ficar mais óbvio, esse Title não é palavra-chave, não é um tag title do HTML. Podia, inclusive, chamar "Titulo". Em vez de Title, vamos deixar Titulo para ver como fica.

Isso não é HTML, isso é algo que o React vai entender e procurar se existe algum componente no escopo com esse nome "Titulo". Se não tiver ele vai dar erro, se tiver ele substitui isso por "Boas vindas de volta".

**Mario:** Exatamente. Só de colocar isso, nós informamos que criamos a nossa tag, colocamos o h1 com o título. Está duplicado aqui. Podemos verificar no navegador que ele parou de aplicar o estilo.

Nesse momento, conseguimos mostrar para a galera como criar o componente, que o estilo do style jsx só aplica no componente a que ele pertence, e que esse Titulo está ignorando qualquer coisa da tag.

O que faltou para conseguirmos pegar esse valor "Boas vindas de volta" dentro da HomePage e jogá-lo no h1 da função Titulo()?

**Paulo:** Agora, de alguma forma, temos que passar como argumento, como parâmetro da função Titulo(), quero pegar essa string e jogar para o h1 dela. Como se fosse argumento1 e argumento2.

function Titulo(argumento1, argumento2) {

return (

<h1> Está ignorando qualquer coisa da tag</h1>

);

}

**Mario:** Isso é uma coisa importante, Paulo. Vejo que muita gente que aprende React decora essa estrutura, não vêem isso como uma função, e sim como um componente do React. É importante falar dessa questão de argumento, porque é justamente o argumento que receberemos no Titulo, que têm as props que são as propriedades que o React nos passa para que possamos trabalhar.

Se eu der um console.log() de props, repare que no console do navegador, para chegar no console clicamos com o botão direito e selecionamos "Inspecionar" e clicamos na aba "Console". E ele traz para nós um objeto.

Object

children: "Boas vindas de volta!"

[[Prototype]]: Object

**Mario:** Isso que aparece à direita no console, "index.js?bee7:3", são coisas internas da ferramenta. Esse "3" é referente à linha 3 do código, foi onde colocamos o console.log().

O valor do props é o children, children tem o valor do "Boas vindas de volta!", que é o que estamos passando aqui no <Titulo> do componente HomePage. Esse valor que vai no miolo da tag é o que chamamos de children. Se pegarmos o props.children em console.log(props.children) conseguimos ver o valor do "Boas vindas de volta!".

Antes era objeto, virou "Boas vindas de volta!", se eu carregar, o navegador mostra só o "Boas vindas de volta!".

Dado que temos o props. children, em teoria, é só passar esse valor aqui no meio da tag h1 de Titulo.

function Titulo(props) {

console.log(props.children);

return (

<h1>props.children</h1>

);

}

**Mario:** Salvei. E voltando no navegador vemos que no lugar de título ele só colocou o texto "props.children". Paulo, pegando seus conhecimentos mais gerais de programação, o que você acha?

**Paulo:** É porque ele não sabe que você está falando "por favor, pegue uma variável". O que você escrever aí nessa tag h1 vai sair como texto, tem que usar um cifrão? Um cerquilha ou algo assim para informar que quero buscar uma variável chamada props e quero pegar o atributo children dela.

**Mario:** Para isso, basta usarmos as chaves, ou, como eu chamo, os "bigodinhos", {},{props.children}. E o valor que estiver dentro dessas chaves é tratado como valor dinâmico do JavaScript. Se você tiver uma variável, um objeto, uma prop, você consegue receber esse valor aqui e trabalhar em cima dele.

function Titulo(props) {

console.log(props.);

return (

<h1>{props.children}</h1>

);

}

**Mario:** Vamos salvar. E voltou a aparecer o "Boas vindas de volta!" no navegador. Mas eu falei que a ideia do componente era começar a juntar as coisas, então podemos trazer esse style para o Titulo também, para começar a agrupar. Então, teremos tanto o componente com o trecho de HTML dele e com o trecho de CSS dele.

O que bate justamente nessa conversa que tivemos sobre o Material UI, ele tem lá o componente calendário, que tem todas as lógicas para montar um calendário, inclusive os estilos dele.

Por isso o pessoal começou a juntar tudo mais próximo uma coisa da outra. A gestão fica mais fácil. Você olha para um pedaço do código e vê que tudo o que você precisa está lá, o HTML, o CSS e o JavaScript.

Vou recortar o trecho do style jsx que está no HomePage() e colar no Titulo().

function Titulo(props) {

console.log(props);

return (

<h1>{props.children}</h1>

<style jsx>{`

h1 {

color: red;

}

`}</style>

);

}

**Mario:** Vai aparecer um erro dizendo "JSX expressions must have one parent element.",ou seja, expressões JSX devem ter apenas um elemento pai. Para resolver esse problema, por enquanto, podemos colocar uma div em volta, assim fica mais fácil de mexer. Agora temos, dentro do Titulo, uma div, um h1 e o style jsx.

function Titulo(props) {

console.log(props);

return (

<div>

<h1>{props.children}</h1>

<style jsx>{`

h1 {

color: red;

}

`}</style>

</div>

);

}

**Mario:** Vou salvar. E o título está novamente com a cor vermelha no navegador. Aí tem outra coisa, no console da página no navegador podemos ver que começamos a "sujar" o HTML, colocamos uma div só para resolver um pequeno problema.

O JSX tem um suporte para isso, ele nos possibilita passar uma tag vazia. Em vez de colocar <div> </div> podemos deixar <> </>. Fica uma tag meio fantasma, mas ela tira esse elemento extra que está em volta e deixar só o h1. Ele só serve para conseguirmos agrupar as coisas mesmo e assim colocar tanto o style quanto o h1 juntos.

Agora, podemos colocar mais estilos, além da cor vermelha podemos passar tamanho da fonte de 24 pixels, font-size: 24px; e peso da fonte de 600, font-weight: 600;.

function Titulo(props) {

console.log(props);

return (

<>

<h1>{props.children}</h1>

<style jsx>{`

h1 {

color: red;

font-size: 24px;

font-weight: 600;

}

`}</style>

</>

);

}

**Mario:** Vou salvar. E, no “Inspecionar” do navegador, vemos que ele está aplicando tudo o que passamos no estilo. Está funcionando corretamente. Paulo, todo título necessariamente vai ser um h1?

**Paulo:** Não.

**Mario:** Faz sentido que a tag desse título possa mudar dependendo de onde vamos colocá-lo?

**Paulo:** Em alguns casos, sim.

**Mario:** O React também nos dá essa flexibilidade. Parece algo de outro mundo, mas basta informar que queremos que, por exemplo, o título seja h2, <Titulo tag="h2">.

function HomePage() {

//JSX

return (

<div>

<Titulo tag="h2"> Boas vindas de volta!</Titulo>

<h2>Discord - Alura Matrix</h2>

</div>

)

}

**Mario:** Se voltarmos no console do navegador, veremos que ele tem agora a children e recebeu uma propriedade tag:"h2". Ou seja, todos os atributos das tags do HTML viram props do seu componente. E com esse valor de tag, posso criar uma variável aqui na função Titulo de const Tag = props.tag; e podemos trocar o h1 pela Tag.

function Titulo(props) {

console.log(props);

const Tag = props.tag;

return (

<>

<Tag>{props.children}</Tag>

<style jsx>{`

h1 {

color: red;

font-size: 24px;

font-weight: 600;

}

`}</style>

</>

);

}

**Mario:** Esse valor de texto que estamos passando passa a trocar qual tag estamos renderizando. Então você pode passar uma div, um h1, qualquer tag que você quiser aqui que ele vai fazer isso funcionar do jeito que esperamos.

Quando copiamos o estilo teve um detalhe que passou despercebido e é muito importante. Esse style jsx, além da prop, ele tem aqui dentro abertura de chaves e tem um recurso do JavaScript que é a template string, um recurso em que você pode vir aqui dentro e passar aquela anotação de cifrão e chaves ${}, ou, como eu gosto de chamar, "ostentação bigodinhos", e passar algum valor dinâmico. Então podemos dizer que essa Tag vai gerar esse estilo também.

function Titulo(props) {

console.log(props);

const Tag = props.tag;

return (

<>

<Tag>{props.children}</Tag>

<style jsx>{`

${Tag} {

color: red;

font-size: 24px;

font-weight: 600;

}

`}</style>

</>

);

}

**Mario:** Então, acabamos de criar um componente de título que funciona com qualquer tipo de tag. Isso aqui é a chave de muita empresa que começa a querer normalizar como a equipe de design fala com o pessoal de código. Não necessariamente o estilo do título da empresa tem a mesma semântica do HTML para todo lugar, então pode ter essa diferença de você querer aproveitar o estilo, mas a semântica ser diferente.

Se eu salvar e voltar para o navegador, ele aplicou no h2 esse estilo. E se mudarmos a tag lá, ele vai mudando as coisas para nós.

Estamos usando os poderes do React para valer agora, temos o Titulo, conseguimos passar qual tag ele deve ter, recebe essa tag aqui em cima no const Tag, passa o children, o conteúdo que queremos renderizar, e gerencia os estilos que essa tag de título que estamos criando vai ter.

Esses estilos não são globais, a fonte do projeto é uma coisa que fica meio global, uma definição mais genérica que temos, o body também tem uma margem, são várias coisas que o pessoal usa o estilo que chamam de reset do CSS.

Para conseguirmos fazer essa parte do reset junto com as fontes, espaçamento e tudo mais, vamos criar o nosso global style. Criaremos uma tag que terá esses estilos globais e vai aplicá-los na página inteira, <GlobalStyle>. Esse GlobalStyle é uma convenção, é como o pessoal chama esse lugar que tem os estilos globais da aplicação. Tudo que for genérico para resetar as coisas e organizar fica nesse GlobalStyle.

Legal que isso bate muito com o conceito de reset de CSS. Inseri a tag <GlobalStyle> na HomePage() e gerei um erro de propósito aqui, para mostrar que sempre que você tentar usar um componente que não foi criado ainda, o navegador vai apresentar esse erro "ReferenceError: GlobalStyle is not defined", ou seja, não criamos ainda.

Vamos criá-lo aqui no início do código.

function GlobalStyle() {

return (

);

}

**Mario:** Como é só estilo aqui, em teoria, que valor retornamos aqui?

**Paulo:** Vamos colocar aquele style jsx?

**Mario:** Exatamente. Para fazer isso passamos também as chaves, a template string e o valor global.

function GlobalStyle() {

return (

<style global jsx>{`

`}</style>

);

}

**Mario:** Se colocarmos que tudo vai ter background da cor preta,\*{background: black}, ele pintou tudo de preto no navegador.

Se comentarmos a tag <GlobalStyle> que está na HomePage, para comentar no JavaScript usamos essa sintaxe das chaves com barra e asterisco, {/\*<GlobalStyle>\*/}, aqui estou usando o atalho "Ctrl + Barra" para inserir esses sinais de comentário.

Para testar enquanto vai fazendo, você pode comentar e descomentar linhas, mudar valor etc. Para se acostumar com o que está acontecendo aqui.

Vou descomentar a linha <GlobalStyle>, e nesse estilo global podemos colocar várias coisas. Podemos resetar o margin que é uma coisa que muita gente faz. E colocar também o espaçamento (padding) de zero, e box-sizing: border-box.

function GlobalStyle() {

return (

<style global jsx>{`

\*{

margin: 0;

padding: 0;

box-sizing: border-box;

`}</style>

);

}

**Mario:** Tem um vídeo excelente da Vanessa da Alura indicando como funciona o modelo de caixa do CSS e porquê muita gente usa isso no reset e tudo mais.

Então, Paulo, isso vai crescendo, colocaremos mais coisas e é um código mais específico. Em cada projeto o pessoal vai criando o seu reset e nós resolvemos trazer algumas coisas para facilitar a vida da galera que está fazendo.

Para não termos que fazer toda essa arquitetura de CSS, porque para fazer uma aplicação não é só sentar e escrever os estilos. Tem bastante coisa a se fazer e vamos trazer coisas que nos ajudarão a fazer esses estilos. E cada pessoa vai poder personalizar, fazer não só o do Matrix, mas fazer um do Pokémon, do Cavaleiros do Zodíaco, o que vocês quiserem. Para isso, vamos pegar o GlobalStyle que foi definido para o nosso projeto. Vou acessar um link que também está na página da Imersão.

**Paulo:** Nós já entendemos o funcionamento básico de um componente. Para fazer uma página bonita como esse layout da nossa página de boas-vindas do Discord Alura – Matrix, tem alguns detalhes de CSS para essa aparência. Nós vamos pegar alguns desses detalhes no código do link que está disponível na página da Imersão. Agora vai ter um pouco de copiar e colar, mas lembrando que tudo o que vamos copiar e colar aqui você já entendeu.

O que você ainda não entendeu pode ser algum detalhe de CSS, de estilo. Já o componente em si é, basicamente, como criamos. Mas agora em vez de apenas título, teremos box, input text, imagem, outros componentes menores. E poderemos trabalhar dessa forma componentizada que vimos na biblioteca do Material UI, que veremos na biblioteca Skynex UI.

**Mario:** No Skynex UI teremos vários desses utilitários de botão, de componente, campo de texto, entre outros. Vai ter o link para você pegar, vai ter vídeo no meu canal. Criamos isso especificamente para você conseguir fazer os estilos de um jeito mais fácil, conseguir replicar e ter seu próprio tema.

Vamos começar a trazer com calma, aos poucos, e colocar no ar também. No final dessa aula colocaremos o projeto no ar, em menos de cinco minutos.

Vamos pegar os estilos no link do GlobalStyle.js. Um dos ajustes que fizemos aqui foi esse App fit Height, para nossa app ficar com tamanho 100%. Quem nunca fez um site em que o rodapé não ficava 100%? Ficava meio feio, no meio da página. Aqui conseguimos corrigir só com CSS.

Vamos copiar e colar no nosso código, no lugar do GlobalStyle().

function GlobalStyle() {

return (

<style global jsx>{`

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

list-style: none;

}

body {

font-family: 'Open Sans', sans-serif;

}

/\* App fit Height \*/

html, body, #\_\_next {

min-height: 100vh;

display: flex;

flex: 1;

}

#\_\_next {

flex: 1;

}

#\_\_next > \* {

flex: 1;

}

/\* ./App fit Height \*/

`}</style>

);

}

**Mario:** Também vamos aplicar essa questão do tema. É uma coisa que às vezes passa despercebido, mas é importante também. Por enquanto, estamos usando a cor vermelha na fonte, mas no layout está branco e pode ser que na sua aplicação você queira deixar com outra cor. Para isso, padronizamos um arquivo que nomeamos de config.json.

**Mario:** este arquivo config.json tem um spoiler do que vai ser a última aula, que são os "stickers" que iremos passar.

Mas também tem a parte de tema com apenas duas paletas de cores: as primárias que definem o verde do botão, e as neutras que definem a caixa, e podemos definir como quisermos.

No coolors.co, apesar dos muitos pop-ups que aparecem, conseguimos gerar novas cores apertando a tecla de espaço, e podemos ter diversos tons das cores clicando em no ícone de "View shades".

Podemos ter diferentes tons de cores usando a paleta que selecionamos, e mesmo quem não é designer pode criar uma página interessante.

Copiaremos o código de config.json, criaremos o arquivo na raiz do projeto com o mesmo nome e colaremos o texto.

Também rodaremos o Format Document usando "Ctrl + Shift + P" ou "Command + Shift + P" para formatarmos o documento.

Com isso, a cor vermelha em color: do index.js deverá vir a partir do config.json. Fazemos isso indo ao topo do arquivo index.js e importando appConfig a partir de './' se fôssemos usar o atual, mas como subiremos da pasta "pages" para a superior, escreveremos '../' para que o próprio VSCode nos ajude a importar as coisas, e depois o arquivo config.json.

import appConfig from '../config.json';

**Mario:** Queremos trazer todo o conteúdo e armazená-lo em appConfig. Ao passarmos o cursor do mouse sobre este caminho, o VSCode indica que está declarado mas ninguém está usando. Então no lugar de color: red, usaremos a sintaxe de cifrão e chaves ou "ostentação e bigodinhos", que receberá appConfig. e, após o ponto, é sugerido os próximos valores possíveis.

Portando, digitaremos appConfig.theme.colors.meutrals., e após o ponto teremos várias opções de valores, que são as escalas de cor muito usadas por designer e empresas.

Escolheremos [000] entre colchetes e salvaremos.

//código anterior omitido

function Titulo(props) {

console. log(props) ;

const Tag = props.tag;

return (

<>

<Tag>(props.children:</Tag>

<style jsx>("`

${Tag} {

color: ${fappConfig.theme.colors.neutrals[000]};

font-size: 24px;

font-weight: 600;

}

`}</style>

</>

);

}

//código posterior omitido

**Mario:** Feito isso, voltaremos ao navegador e teremos um erro "Legacy octal escape is not permitted in strict mode", mas para resolver esse problema, basta colocar este valor numérico da cor como string da chave .neutrals['000'] entre aspas simples.

Feito isso, atualizaremos a página e, ao abrirmos o inspetor de código, veremos a cor branca. Se colocássemos '900' por exemplo, já seria uma cor cinza bem escuro.

Podemos trabalhar com isso, mas agora há vários componentes para criarmos.

**Paulo:** Isso será para refletirmos a página que queremos

**Mario:** Para ganharmos tempo e focarmos em React, também passaremos essa base do template da página, a qual você pode alterar como quiser, e já fornecemos um suporte todo integrado com a biblioteca SkynexUI no Storyboard.

**Paulo:** Vale lembrar que esta lib foi criada e desenvolvida por brasileiros, principalmente o Mario Souto.

**Mario:** Exatamente, e ela foi toda pensada para simplificar o acesso mesmo, então possui suporte para ícones, podemos mudar para ícone do Ebay, Facebook, entre outros.

Conforme escolhemos as opções, a biblioteca vai gerando o componente com as configurações, então podemos copiar o código e explorar os recursos.

O mais interessante é que ela usa tudo o que já fizemos, em que cada componente é a mesma coisa que fizemos em nosso Titulo(). Portanto, recebe até o CSS dinamicamente.

Por exemplo, podemos ir ao componente React HomePage e dizer que temos um style,o qual receberá backgroundColor: como 'black' para termos um fundo preto.

//código anterior omitido

// Componente React

function HomePage() {

// JSX

return (

<div style={{ backgroundColor: 'black'}} >

<GlobalStyle />

<Titulo tag="h2">Boas vindas de volta!</Titulo>

<h2>Discord - Alura Matrix</h2>

</div>

)

}

export default HomePage

**Mario:** Salvando desta forma e atualizando a página no navegador novamente, veremos que o fundo ficou preto. Porém, o style não nos permite trabalhar com resoluções da tela, apenas com media queries com tela pequena, tela grande, tela de celular e etc.

Mas se formos na documentação do SkynexUI no Storyboard e acessar o componente "Box Component" na lista lateral, poderemos notar que, se passarmos uma propriedade chamada styleSheet como uma "folha de estilo".

Na verdade, podemos passar tanto seu valor quanto às resoluções. Podemos alterar os nomes das cores em cada um dos atributos da forma como quisermos.

No Storyboard, podemos clicar no ícone de resolução de tela na aba superior e testar os diferentes tamanhos de exibição.

**Paulo:** Encontramos o link do Storybook aqui, com a documentação dos componentes. É uma live em que podemos pegar os componentes e testá-los.

**Mario:** Exato. Também deixarei um vídeo no canal aqui, para que as pessoas possam criar a documentação no Storybook corretamente. Perceberemos que tudo que mudamos foi refletido no texto, então se quisermos copiar, será exatamente o código do tema configurado.

Então é muito prático para termos autonomia dentro de um time. Particularmente, gosto muito dessa biblioteca e fico feliz em compartilhar tudo o que usamos até agora.

Quando clicamos no valor da tag, aparece e descrição de quais tags HTML podemos passar para os componentes, então está pronto para uso!

Para isso, temos que fazer a instalação. Iremos novamente ao VSCode, abriremos o terminar e faremos "Ctrl + C". Pararemos o processo e rodaremos yarn add @skynexui/components.

Se acessarmos este [link do GitHub](https://github.com/skynexui/components), encontraremos vários elementos interessantes. A Vercel é patrocinadora desse projeto, então libera o acesso.

A Alura também é sponsor e acredita neste prometo, e poderemos copiar o comando tanto via yarn quanto via npm. Há todos os porquês para podermos avançar.

Lembrando que estamos livres para usarmos o que quisermos, então por mais que tenhamos essa versão usando o styled-jsx, também poderíamos usar o styled-components, mas a questão é que teremos que fazer todos os componentes manualmente, oq eu daria muito mais trabalho.

Mas no próprio GitHub da Vercel neste link, encontraremos vários exemplos, e inclusive o pull request aberto para exercitarmos essa lib que estamos usando.

Ao clicarmos em "with-styled-components", teremos a mesma estrutura base que aprendemos a criar no caminho, então não teremos mais dúvidas. Encontraremos o package.json, os scripts, as dependências, o que tem a mais e a menos e etc. Assim teremos mais autonomia para trabalhar da melhor forma.

Depois de instalarmos no terminal do VSCode, iremos ao index.js de "pages" e faremos o import a partir de '@skynexui/components'. Teremos vários componentes, como a caixa, o botão, ícone, imagens, campos de texto e etc. Como estamos fazendo um por um, importaremos o Box, Button, Text, TextField, Image.

import { Box, Button, Text, TextField, Image } from '@skynexui/components';

import appConfig from '../config.json';

//código posterior omitido

**Mario:** Já mais adiante para a parte do miolo, comentaremos o que fizemos na função HomePage() para não termos um erro por enquanto, e abaixo deste colaremos a estrutura inteira a partir de PaginaInicial() disponível neste link.

O export default define o padrão do arquivo que será a página. Formataremos o documento, chamaremos yarn dev no terminal e abriremos no navegador novamente.

Com isso, teremos um Server Error dizendo que o Title não está definido, então mudaremos para <Titulo>.

**Paulo:** É verdade, nós mudamos. Mas quem quiser pode mudar também.

**Mario:** Exatamente, por isso temos o link com tudo corrigido disponibilizado para as alunas e alunos.

Porém, na função Titulo(), poderemos colocar qualquer tag, mas não há uma tag padrão, e temos que indicar isso. Então, caso não tenha nada em props.tag, adicionamos || para pegar o valor padrão caso haja, e se não houver, pega o 'h1'.

//código anterior omitido

function Titulo(props) {

const Tag = props.tag || 'h1';

return (

<>

<Tag>{props.children}</Tag>

<style jsx>{`

${Tag} {

color: ${appConfig.theme.colors.neutrals['000']};

font-size: 24px;

font-weight: 600;

}

`}</style>

</>

);

}

//código posterior omitido

**Mario:** Com isso, teremos tudo funcionando corretamente. Teremos nosso título e h1 por padrão, mas se formos até o <Titulo> que acabamos de corrigir e passarmos a tag como "h2" por exemplo, o programa irá mudá-lo, pois só pega o valor de 'h1' caso nada passe um valor para props.tag. Desta forma, podemos ter a versão do projeto rodando sem problemas.

Mas se formos para config.json e mudarmos a cor primária de "500": para "red", veremos a diferença interessante na versão vermelha da nossa página.

Podemos usar qualquer cor, mas o importante é fazer o projeto customizado e nos mandar para colocarmos no ar.

**Paulo:** Perfeito. Lembrando, a tag <Box> é da biblioteca que estamos usando, e ela recebe como parâmetro várias coisas de CSS que podemos mandar. Quem está fazendo pode não saber o que é cada elemento, então podemos até deletar para ver o que acontece e entender melhor.

A parte de CSS não é nosso foco, mas já estamos entregando com configurações bem bonitas para seu portfólio. Mas claro, quem domina CSS pode explorar mais as possibilidades.

Quem não tem domínio, não precisa se preocupar com essa parte de código dos componentes do React, e sim apenas a estilização, como atributos do photoshop ou Figma para customizar.

Reepararemos que tudo está em <Box>, e se deletarmos o styleSheet ficará feio, mas o que queremos irá funcionar.

**Mario:** Exato, se fizermos isso voltarmos ao navegador, veremos as diferenças visuais na página, mas as funcionalidades estão funcionando. Com isso, as pessoas poderão ter algo mais apresentável no portfólio.

Também é interessante assistir a série "Pare de chutar CSS" com Marco Bruno neste link para melhorar as habilidades.

Não há nenhuma magia, é CSS da mesma forma com as tags normal, e cada <Box> virará uma <div> por padrão, então podemos ir ao navegador e inspecionar cada um deles.

Como são <div> normais, podemos passar outra tag para termos tudo certo para colocarmos o que quisermos.

Terei o meu usuário por padrão, mas nas próximas aulas, ao trabalharmos mais com estado, campos de textos e outras customizações, podemos mudar.

Por enquanto já colocaremos no ar, pois já passamos bastante conteúdo nesta aula. Para isso, colocaremos no site da empresa por trás do Next.js, que é a Vercel.

Na [Home da Vercel aqui](https://vercel.com/home), iremos nos familiarizar. A proposta é facilitar o trabalho e colocar o projeto no ar o mais rápido possível.

Para isso, a única coisa que temos que fazer é subir o código no GitHub. Então criaremos um novo repositório chamado "aluracord-matrix", mas podemos colocar qualquer outro nome que quisermos.

O nome do projeto é importante para montarmos a vitrine, mas explicaremos melhor. Deixaremos público, mas por enquanto ficará privado para as pessoas não verem antes de terminarmos a aula.

Criado o projeto, há o tutorial do Git com git init. Então copiaremos isso, fecharemos os arquivos no VSCode e escreveremos o comando no terminal.

Ao executarmos, receberemos uma mensagem dizendo que há muitos arquivos, e nos pergunta se queremos colocar no .gitignore. Fecharemos a mensagem e perceberemos que está monitorando cerca de dez mil arquivos no ícone lateral.

Para resolvermos isso, faremos o comando npx gitignore node no terminal para gerarmos o arquivo para o git, que é essa ferramenta de compra de versão que irá monitorar o arquivo, gerenciar as versões e outras tarefas, a qual irá ignorar tudo o que tiver dentro deste arquivo .gitignore.

**Paulo:** esse npx já veio com o Node.js quando instalamos.

**Mario:** Exato, está por padrão. Se instalarmos o Node.js, teremos node -v, o npm -v e o npx junto, o qual é o atalho para o npm rodar alguma coisa mais remota sem instalarmos na máquina.

Na aba de "source control" do git, podemos clicar na seta circular para recarregar. Com isso, mostrará somente cinco arquivos ou mais, se tivermos criado mais.

Clicamos em "Staged Changes" na aba lateral do VSCode, que é equivalente ao comando git add . no terminal. Fazendo o refresh, exibiremos "Staged Changes", que é justamente o próximo comando que o GitHub exibe no tutorial.

Em seguida, teremos o git commit para criarmos um pacote com todas as orientações. Inclusive, há uma série "Git e GitHub para sobrevivência" aqui na plataforma Alura comigo, Mario Souto, aprofundando no assunto para quem se interessar. Há muito conteúdo avançado, mas as primeiras aulas são básicas.

Depois, teremos o comando git branch para renomearmos caso tenhamos nossa Branch Master ou algo do gênero. No caso já está como main. Após isso, colocaremos o git remote add origin no terminal para vincularmos nossa pasta local com a remota do GitHub, mas em breve entenderemos o porquê de "pasta".

Por fim, escreveremos o git push para "empurrar" nosso código para a pasta remota. Nesta mesma página do GitHub, clicaremos no ícone de engrenagem ao lado do título "About" no canto superior direito para abrirmos a janela de "edit repository details".

No campo "Description" colocaremos a descrição do projeto, como "Projeto criado na Imersão React com a @alura, @omariosouto e o @peas".

Já em "Topics", adicionaremos "aluracord", "imersao-react" e várias opções para marcarmos e exibirmos no "About".

Para finalizarmos, iremos publicar. Portanto, iremos à home da Vercel, faremos o login em que basta ter uma conta no GitHub para acessar, abriremos o dashboard, clicaremos em "New Project", depois em "aluracord-matrix" para importarmos apertando o botão de "Import" ao lado do projeto, e irá aparecer automaticamente.

Com isso, chegaremos em "Configure Project" onde poderemos alterar o nome do projeto se quisermos. Por fim, clicaremos em "deploy" e pronto.

Aguardaremos o build e acessaremos a parte com título de "Congratulations!" indicando que publicamos com sucesso, e em que poderemos ver o nosso trabalho exibilido na tela.

Ao clicarmos em "Go to Dashboard", veremos a url do projeto, e inclusive poderíamos comprar um domínio, mas por enquanto conseguimos cumprir nosso objetivo.

Nas próximas aulas, preencheremos alguns campos, faremos os botões funcionarem corretamente, entre outras ideias.

Agora é o momento de compartilhar conosco, pois queremos ver as criações para fazermos uma exposição de projetos.

**Paulo:** Teremos uma live para mostrar os projetos mais interessantes. Como ainda criaremos o chat e tudo irá se interligar, veremos que o projeto ficará incrível!

Não somente em questão visual, mas também de funcionalidades. É justamente o que queremos, afinal o Desenvolvimento em T se aprofunda em front-end mas terá noções de back-end e UX.

Na próxima aula, começaremos repassando este código para revisarmos, e em seguida partiremos para as funcionalidades e criação de outra tela.

**Mario:** Sim, estruturaremos essa parte para entendermos bem, aprenderemos a criar novas páginas com algumas novas configurações, e avançaremos para a tela do chat, rodaremos o chat offline para depois fazermos integração com a parte remota, entre outros processos para completarmos o melhor projeto possível.

**Paulo:** As próximas aulas serão mais curtas pois fomos bastante minuciosos nesta, e neste momento final vimos bastante cola de CSS, e inclusive as tags podem ser alteradas e criadas.

**Mario:** Exatamente! O <Box> é uma <div>, e o título já é uma caixa também, pois podemos passar qualquer tag, basta criarmos.

Algo que aprendemos bastante é tentar copiar as libs que usamos para aprendermos mais sobre as ferramentas.

**Paulo:** Perfeito, estamos aguardando vocês no Discord para conversarmos e compartilharmos ideias juntas e juntos, afinal queremos que vocês participem do Aluraverso.

A Alura se expande cada vez mais com pessoas de diferentes profissões que aprenderam muito conosco, o que veremos bastante mais adiante.

Estou muito animado de trazer mais uma Imersão React para vocês, que é um framework muito importante. Mas também vamos além, e mostramos Nex.js, JavaScript, conceitos por trás, e não só copiando e colando mecanicamente.

Isso também é importante, mas queremos aprofundar mais para que vocês possam conhecer mais sobre como a Alura funciona e pensa os cursos e formações.

Assim, conseguirmos ter a experiência em grupo de uma verdadeira escola. Portanto, até breve!

**Mario:** Até a próxima aula!

**Mario:** Olá, pessoas!! Boas-vindas à **segunda aula** da **Imersão React** da **Alura**. Estamos aqui, eu, **Mario Souto**, e o **Paulo Silveira**. Paulo, o que temos para hoje? Você pode inciar o assunto da nossa aula?

Mario é um homem branco, de bigode, cavanhaque e cabelos curtos na cor castanho escuro. Ele veste uma blusa preta e usa headphones também na cor preta. Está sentado em uma cadeira gamer, com um microfone próximo ao rosto. Ao fundo, é possível ver uma estante com cds e alguns bonecos sob uma luz rosa do lado esquerdo e outra azul do lado direito.

Paulo Silveira é um homem branco, de barba e cabelos curtos na cor castanho escuro. Ele usa óculos e está vestindo uma camiseta branca. Há um microfone que sai da lateral direita e se aproxima do seu rosto. Ele está sentado em frente a uma prateleira com alguns livros e objetos, como um quadro do Hipsters Ponto Tech, uma planta e uma miniatura da escultura "O Pensador".

**Paulo:** Pois é, Mario, nós tivemos uma aula muito potente, com vários passos até chegar ao deploy e colocar no ar, no Cloud. Algumas pessoas usaram outros clouds, mas já temos muitos projetos no GitHub. Se você for ao GitHub e procurar por **#ImersãoReact**, notará que muitas pessoas colocaram.

Inclusive, se você não colocou, te convidamos a editar o seu projeto e incluir a hashtag da Alura e da Imersão React, porque temos muitas atividades para fazer nessa semana, são muitos conteúdos, exercícios e desafios extras que, ao final dessa aula, já começaremos a passar.

Na primeira aula tivemos muitos códigos e informações. Nessa, teremos também desafios para que você consiga melhorar o seu projeto e ir além do que propomos. No método da Alura, temos roteiro, cursos, e desejamos que você personalize, "dê a sua cara", para o seu projeto e para o curso, criando portfólio, colocando no GitHub, mostrando quem você é.

Nós chamamos isso de "vitrine de Dev", é uma forma de mostrar como você aprende, pensa e sua capacidade de ir além. É esse profissional que as empresas estão buscando.

Eu estou no mercado de educação, tecnologia e contratação há mais de uma década e percebo que, cada vez mais, as empresas estão propagando essa ideia, porque a tecnologia muda, o React daqui a cinco anos será outro, já existe o Next.js, que estamos estudando também, e isso tudo mudará, a versão mudará. Tendo este contexto eu e as empresas queremos saber se você é uma pessoa preparada para aprender e reaprender.

Para começarmos a aula, eu queria mostrar, Mario, que a Alura está muito além da Imersão, dos cursos e da formação. Existe uma página, que chamamos carinhosamente de **AluraVerso**, é o multiverso da Alura. Nós somos mais que uma plataforma, somos uma escola. No AluraVerso é possível encontrar todos os canais, influenciadores e cientistas que patrocinamos. Futuramente incluiremos as ONGs e os grupos minorizados que patrocinamos e ajudamos.

Portanto, no AluraVerso há algumas dessas empresas e pessoas que estamos muito perto, por exemplo, Átila Iamarino, BrazilJS, Manual do Mundo, são pessoas que ajudam a ciência no Brasil e no mundo e eu gostaria de mostrar que você pode ir muito além conosco. Muito além da escola.

Agora, vou aproveitar a deixa, começou um barulho de furadeira na minha casa, para que você, Mario, recapitule o momento em que paramos e explique quais os próximos passos.

**Mario:** Perfeito, Paulo. Inclusive, é uma honra fazer parte do AluraVerso e estar aqui, passando esse conteúdo de React que eu sei que ajuda muitas pessoas a conseguirem mudar de emprego ou até mesmo o primeiro emprego. Enfim, é uma honra imensa. Vamos revisitar o nosso projeto, que publicamos durante a última aula.

Estou com o Terminal aberto, mostrando que estamos na aula 2, aluracod git:(aula2). Não mudei nada no código, apenas a branching no GitHub para termos a main. Estou na aula 2 agora. Você pode seguir na main, não é obrigatório quebrar a branching. Vamos rodar o yarn dev para subirmos o nosso projeto e revisitarmos um pouco da última aula, contextualizando o que faremos agora.

Vamos abrir uma aba nova, acessar [localhost 3000](http://localhost:3000/). O usuário do Paulo já está habilitado e aberto. Em "Explorer > Open Editors", dentro da pasta "pages" e nela o arquivo "index.js", no VSCode, temos os nosso GlobalStyle, que vamos minimizar apertando a seta à frente de function GlobalStyle() {. Vamos minimizar o título também, function Titulo(props) {.

Mais abaixo, temos o nosso código base e, ao final da aula, colocamos uma página inicial customizada, export default function PaginaInicial() {, também tínhamos o usuário do Paulo no começo, const username = 'peas'; e o GlobalStyle.

export default function PaginaInicial() {

const username = 'peas';

return {

<>

<GlobalStyle />

<Box

Vários boxes são divs. Um deles tem o as="form", que está mudando algo como a tag dele, um box como formulário. Em seguida, o styleSheet={{ com todos os estilos que estamos passando. Mais abaixo, existe a área da foto, que comparamos no GitHub. Inclusive, fica esse segredo: se você pegar qualquer usuário do GitHub e colocar .png no final, você consegue pegar a imagem .png da pessoa que você segue ou do seu próprio usuário.

<Image

styleSheet={{

borderRadius: '50%',

marginBottom: '16px',

}}

src={'https://github.com/${username}.png'}

/>

Para isso, basta acessar o GitHub, por exemplo, vamos até o meu usuário acessando a URL <https://github.com/omariosouto>. Se ao invés de barra, eu colocar ".png", isto é, <https://github.com/omariosouto.png>, ele traz a minha foto. Fica esse easter egg do GitHub para vocês.

Portanto, nós temos todo esse trecho de código. Vamos pegar o username que aparece acima, atribuir na variável na imagem e do texto. São conteúdos que já estudamos na última aula, usar o "bigodinho" e atribuir variáveis.

<Text

variant="body4"

styleSheet={{

color: appConfig.theme.colors.neutrals[200],

backgroundColor: appConfig.theme.colors.netrauls

padding: '3px 10px'

borderRadius: '1000px'

}}

>

{username}

</Text>

</Box>

{/\* Photo Area \* /}

</Box>

**Paulo:** Nós estudamos o componente do título que era passado como argumento.

export default function PaginaInicial() {

const username = 'peas';

Agora estamos separando uma variável hardcoded, a username, e estamos substituindo essa variável - que é o meu login no GitHub - em dois lugares. Um deles é na imagem, e temos um truque famoso que é usar github.com/ e o seu username.png. Ao entrar no site, ele mostrará a imagem. O username também está aparecendo em um Text. Ele é um componente da biblioteca.

Nós poderíamos usar um componente da biblioteca Material-UI do android ou de outra biblioteca. Nós estamos usando uma nossa, mas cada pessoa pode criar seu próprio componente, por exemplo, "textobonitinho". Então, você cria a function "Textobonitinho" e faz, assim como criamos ontem, a function "Título".

Tudo que estamos usando, nós estudamos anteriormente. O que aparece grande e pode assustar um pouco pela quantidade é o TextField, fullWidth, textFieldColors={{. Essa é a parte que mais tem a ver com a estilização, com o CSQS.

**Mario:** Quase todas as estilizações que vocês estudarão, nos componentes como o Button e TextField, são sempre de cor, por exempo, mainColorLight: appConfig.theme.colors.primary[, porque ele já tem algum tamanho específico: pequeno, médio ou grande. E nas caixas, existe a especificação, não do destino, mas de espaçamento, de display, que compete muito mais à parte estrutural da página.

Então, todo box tem estrutura e todo componente tem a parte visual. Essa parte está bem separada no código, mas o importante, Paulo, é fazermos o username ser alterado quando digitarmos no campo de login (No localhost 300 temos: "Boas vindas de volta! - Discord - Alura Matrix". Abaixo um campo de login seguido do botão "Entrar"). Essa será a nossa meta.

Como fazemos esse processo todo acontecer? Vamos ao nosso código. Nele temos a PaginaInicial() e o TextField. Nós comentaremos o TextField, {/\* <TextField, e adicionaremos a tag input básica do navegador.

**Paulo:** Essa é HTML, não é? Não que não pudesse ser de outro jeito.

**Mario:** Isso! Só para conseguirmos trabalhar garantindo a quem está com um pouco de receio do componente que perceba se tratar da mesma coisa e que conseguimos fazer os dois funcionarem sincronizados.

<input type="text" />

{/\* <TextField

fullWidth

textFieldColors={{

Ao voltar à página (localhost 3000), tivemos um input meio "feio", mas funcional. Se digitarmos o username no campo de login, ele está aparecendo. Gostaria de perguntar sobre o input, Paulo, você se lembra, no tempo em que fazia as integrações do Java para HTML, qual era a propriedade com que definimos um valor inicial para um campo?

**Paulo:** É o value?

**Mario:** Sim! É o value.

**Paulo:** Nossa! Eu não faço isso há uma década!

**Mario:** Não tem problema! Em qualquer campo, se adicionarmos um value padrão, por exemlo, o usuário do Paulo, o peas

<input

type="text"

value="peas"

/>

Ao retornar à página, notaremos que "peas" já aparece no campo de login. Ou seja, o value apareceu. A nossa meta será: sempre que digitarmos, mudar tanto a foto que pegaremos do GitHub, quanto o texto que aparece abaixo da foto, "peas". Essa é a nossa primeira missão.

No centro da tela há um retângulo. Dentro dele, à esquerda, aparece a mensagem: Boas vindas de volta - Discord - Alura Matrix, com um campo de login abaixo e o botão "Entrar". À direita, uma imagem recortada em formato circular do rosto de Paulo Silveira, ele é um homem branco, usa óculos de armação quadrada na cor marrom, tem barba e cabelos curtos e usa uma touca cinza.

Basicamente, teremos que digitar e fazer funcionar. Mas, Paulo, repare que "bizarro", estou digitando no campo de login e nada acontece, isso se dá porque quando colocamos o value no input e no React, ele faz parar de funcionar.

Se retornarmos ao código e apagarmos o value="peas" e dermos um reload (recarregarmos a página) para digitar, ele permite de novo. A reposta disso está no Console.

**Paulo:** Não era esperado funcionar assim.

**Mario:** Não era esperado. No HTML normal, se você escreve o value, ele te deixa digitar. Vamos abrir o Console e verificar o que está ocorrendo. Estamos digitando no campo de login da página. Se adicionamos o value="peas" ao código, no Console aparece uma mensagem de erro: um componente está mudando um input que não está sendo controlado. Isso está causando algo como o value trocar de undefined para um valor definido, o que não deveria acontecer.

Portanto, ele está alertando que deveríamos ter um componente controlado ou um não controlado. Nós discutimos bastante o conceito de componente. Ele é "quem manda". Quando estamos trabalhando com interfaces, o componente é quem ajuda a montar. Porém, a chave da popularidade do React vem de um outro conceito chamado de **estado**.

Para o React, cada vez que digitamos no campo de login, é como se ele estivesse tirando uma nova foto de como está o código no exato momento em que digitamos. É como a água, encontrada nos estados sólido, líquido e gasoso. No código, cada alteração que fazemos é um novo estado. Cada mudança, cada menu que abrimos e fechamos, ou até cada letra que digitamos, é uma alteração de estado.

Quando o React fala de controlled e uncontrolelled é porque ele sabe que se o input tem o valor de value associado a ele, provavelmente esse valor muda. E se ele muda, é necessário dizer ao React que esse valor tem a possibilidade de interagir com esse estado, de gerar uma nova versão. Precisamos de alguma forma dizer que esse valor pode ser alterado mesmo que definamos um valor que está estático, por exemplo, value="peas".

Fez sentido essa ideia de poder mudar? Então, que o React precisa saber de cada caractere que mudou? Por enquanto, ele não sabe. Só escrevemos um texto direto. Para ligar essa parte de mudança de estados e gerar uma nova versão com o React, vamos vincular o value com alguma variável. Se vamos mudar para algo que varia, o caminho lógico é usarmos uma "variável".

Podemos colar a variável usarname, que aparece mais acima no código, no valor que está mais abaixo. Seguiremos a seguinte lógica: abrimos chaves (também conhecidas como "bigodinhos") e chamamos o username.

<input

type="text"

value={username}

/>

Se voltarmos na página e carregarmos, verificando em paralelo o Console, notamos que ainda aparece o erro. Porém, se alterarmos o código em const username = 'peas'; para const username = 'omariosouto';, ele passa a pegar o meu usuário. Na página, perceberemos que está sincronizado o nome com a imagem e o campo. O erro, no entanto, ainda aparece.

Para que o erro pare de aparecer, precisamos recordar que ele mostrou uma mensagem, dizendo: você proveu uma propriedade value para um campo de formulário sem um onChange handler. Significa que passamos a variável, então o React sabe que esse valor vai variar, mas, além do valor da variável, ele precisa saber o que acontece quando ele vai capturar essa foto da página.

Ou seja, ele precisa acompanhar e saber o que fazer cada vez que o usuário digitar. Para isso, ele pede o valor de onChange. Portanto, vamos incluir esse onChange handler no nosso código, que é, basicamente, uma função. Precisamos ir até o input, incluir onChange e passar uma function.

O nome pouco importa, vamos colocar handler apenas para seguir a nomenclatura que geralmente colocam, mas, é possível passar sem nome e vai funcionar da mesma maneira.

<input

type="text"

value={username}

onChange={function handler() {

}}

/>

Também vamos colocar console.log('usuario digitou').

<input

type="text"

value={username}

onChange={function handler() {

console.log('usuario digitou')

}}

/>

Em seguida, carregaremos a página de novo. Ao conferirmos o Console, notaremos que parou de dar erro. Nesse momento, resolvemos o problema. Se digitarmos no campo de login, o React agora sabe que o usuário digitou, sabe também que o valor deveria variar, mas não sabe como mudar esse valor. Fez sentido, Paulo?

**Paulo:** Fez sentido!

**Mario:** Nós conseguimos colar as coisas de maneira que o React sabe do valor, sabe que está capturando a digitação do usuário, mas ele ainda não sabe como muda. Se temos uma variável username, o que deveríamos fazer todas às vezes em que o usuário digita?

**Paulo:** Trocar a variável.

**Mario:** Sim! Trocar a variável, queremos // Trocar o valor da variável. Mas a pergunta é, // Onde ta o valor?.

**Paulo:** Com certeza, vai ter que receber como parâmetro.

**Mario:** Exato! Esse valor vem da ação do usuário, logo, a função onChange só é executada quando o usuário digita. Toda vez que digitamos, ele chama essa função de novo. Coloca parênteses no final e executa.

**Paulo:** Mario, basicamente, o que você está nos mostrando é o funcionamento mais importante do "coração" do React. É quando você quer disparar que outra coisa aconteça em outro lugar. Algo assim.

**Mario:** Exatamente. É isso mesmo. Sempre que digitamos, o navegador nos dá uma variável chamada evento ou infosDoEvento, ou ainda, event. Pode chamar da maneira que você preferir, o nome pouco importa. O importante é a ordem, o primeiro parâmetro terá o evento. Se dermos um console.log() nele, você perceberá que ele é um tanto "bizarro".

<input

type="text"

value={username}

onChange={function (event) {

console.log('usuario digitou', event);

// Onde ta o valor?

// Trocar o valor da variavel

}}

/>

Cada vez que digitamos, ele mostra no Console o SynteticBaseEvent, com bubbles, cancelable, currentTarget, enfim, uma série de informações, e dentre elas há um valor chamado de target com value. Dentro dele, há o valor atual do campo que estamos digitando. No caso, "omariosouto".

target: input

value: "omariosouto"

**Mario:** Nesse caso, ele está pegando o omariosouto. Vamos tentar acessar o event.target.value.

**Paulo:** Aí eu não preciso usar um cifrão, porque está dentro de um bloco Javascript puro, certo?

**Mario:** Exatamente. Se você passa o mouse sobre o event, o VSCode inclusive mostra:

(parameter) event:

ChangeEvent <HTMLInputElement>

Já sobre o .target, o VSCode mostra:

(property)

ChangeEvent<HTMLInputElement>.target

EventTarget & HTMLInputElement

E no value:

(property) HTMLInputElement.value:

string

**Paulo:** Nossa, ele sabe todas essas informações?

**Mario:** Sabe, está bem especificado. Vamos salvar. Repare que quando eu digito a letra "a" na caixa de texto, ela é exibida no final da string que é mostrada no console, por exemplo "omariosoutoa". Se eu digitar "b", essa é a letra exibida, resultando em "omariosoutob".

Isso acontece pois o navegador está tentando colocar mais um caractere ao final do nosso input, mas o React entende que precisa preservar o estado antigo da página, bloqueando essa alteração. Dessa forma, a última letra na string exibida no console vai mudar, mas o que está acontecendo na página não.

O React precisa desbloquear o navegador para que uma nova versão seja gerada e a página atualizada. Nós já temos o valor do campo e uma variável, e agora precisaremos usar um recurso do React.

O valor de username não poderá mais ser somente uma variável, mas também precisará ser capaz de ser alterado internamente de acordo com o estado do React. Para trabalharmos isso, vou comentar a variável username e recriá-la.

Antes de declarar a variável, usarei a instrução React.useState().

export default function PaginaInicial() {

const = React.useState('');

return {

//...

}

Feito isso, o React será importado no cabeçalho da página. Na declaração da variável, ao invés de passarmos somente username, que é o nome dessa variável, usaremos também setUsername, uma função que permite definir um novo valor a essa variável.

export default function PaginaInicial() {

//const username = 'omariosouto';

const [username, setUsername] = React.useState('');

return {

//...

}

**Paulo:** Essa é uma sintaxe do Javascript na qual o useState() devolve dois valores, certo?

**Mario:** Isso. Parece magia, mas vou mostrar para deixar mais claro. Se declararmos uma variável stateDoReact recebendo a chamada de useState() e executarmos o console.log() dessa variável, teremos:

export default function PaginaInicial() {

const username = 'omariosouto';

const stateDoReact = React.useState('');

console.log('stateDoReact', stateDoReact)

return {

//...

}

O retorno no console será um array na qual o primeiro item é um valor vazio e o segundo uma função. Se passarmos o valor omariosouto na chamada de useState(), o primeiro valor do array será omariosouto e o segundo uma função - no caso, a função que altera esse primeiro valor, ou seja, faz o "set".

Como é um array, podemos "desestruturá-lo", pegando o valor de username e o de setUsername. Por convenção, sempre utilizamos essa sintaxe parecida com a de array - [username, setUsername] -, mas expandindo o resultado daquilo que vem depois do igual.

export default function PaginaInicial() {

//const username = 'omariosouto';

const [username, setUsername] = React.useState('');

return {

//...

}

**Paulo:** Preste atenção, pois essa parte da explicação é muito importante, já que é algo que você fará o tempo inteiro em React. Você dificilmente trabalhará com variáveis "puras" como já estávamos acostumados, e bem mais com useState(), com o qual você inicializa uma variável - às vezes até vazia - e recebe o valor que será exibido e quem deverá ser chamado quando for necessário alterá-lo.

Quando você alterar esse valor, o React acionará diversos mecanismos internos para atualizar o que for preciso. É o que chamamos de "observer" (ou "listener"), entidades responsáveis por acompanhar os estados dessas variáveis e fazer as alterações devidas, como alterar uma cor, ativar uma notificação e assim por diante. Às vezes um único evento, como alterar uma simples variável, dispara diversas ações na tela ou no próprio backend.

**Mario:** É verdade, coisa que nós faremos também, por exemplo um chat cuja tela atualiza sempre que há uma nova mensagem.

Continuando, temos o nosso username. Se eu recarregar a página, não teremos nenhum erro e o usuário continua funcionando. Ainda não é possível alterá-lo, já que não utilizamos a função setUsername(), responsável por alterar o valor e gerar um novo estado da página.

Vamos acessar o campo <input> e inserir const valor = event.target.value; em nosso onChange.

<input

type="text"

value={username}

onChange={function {event} {

console.log('usuario digitou', event.target.value);

// Onde ta o valor?

const valor = event.target.value;

//Trocar o valor da variavel

}}

}

Em seguida, para trocarmos o valor da variável, chamaremos setUsername() passando o novo valor.

<input

type="text"

value={username}

onChange={function {event} {

console.log('usuario digitou', event.target.value);

// Onde ta o valor?

const valor = event.target.value;

//Trocar o valor da variavel

//através do React e avise quem precisa saber

setUsername(valor);

}}

}

Eu gosto de deixar esses comentários, e recomendo que vocês deixem também para que fique evidente o que está acontecendo e por que. Isso ajuda bastante a analisar o código no futuro e estudar.

**Paulo:** Isso é mais do que trocar o valor da variável, é trocar por meio do React e avisar quem precisa ficar sabendo, certo?

**Mario:** Exatamente.

**Paulo:** Se eu pegasse aquela variável username = valor, não iria funcionar - primeiramente porque é um const, e também porque o resultado não seria o esperado. É fundamental utilizarmos essas funções do tipo "set".

**Mario:** Sim, sempre vai ser "set" seguido do nome da variável que estamos alterando. Vou salvar as alterações e carregar a página. Agora nós conseguiremos apagar o texto do input, o que também alterará a imagem que é carregada. Se digitarmos "omariosouto", aparecerá minha foto; se digitarmos "alura", será exibido o logo da Alura, e assim por diante.

Isso acontece justamente por essas questões que o Paulo comentou. Conforme alteramos o campo input, o React muda o estado da página em tempo real. Vou inspecionar a página e acessar o código-fonte, pois acho que demonstra bem o que está acontecendo.

Nele, encontramos o nosso <input>, o <img> e o <span>. Repare que, conforme alteramos o que é digitado no input, somente os campos que possuem a variável são alterados - nosso componente de imagem e o campo de texto com o username.

O React é inteligente ao ponto de alterar somente aquilo que é necessário na página, o que costuma ser chamado de "virtual dom". Ele mantém uma cópia, que é aquilo que está na página, e as alterações são feitas em uma versão duplicada dessa página na memória, alterando o que é necessário e comparando as diferenças entre o que está no HTML e o que foi editado, aplicando essas diferenças.

Toda alteração que fazemos na página tem um custo muito grande para o navegador, e o React otimiza esse processo gerando um lote de alterações e aplicando-o de uma só vez. Por isso o React é famoso por sua performance, já que consegue agrupar várias alterações para realizá-las de uma vez só. Isso pode ser visto claramente pelo código-fonte da página, onde todos os elementos são alterados ao mesmo tempo.

**Paulo:** Perfeito. Isso é algo que, se a gente fosse fazer usando Javascript puro, teríamos que alterar em vários pontos do código, talvez até fazer um for.

Aqui as coisas estão "invertidas": o React sabe os pontos em que username é utilizado e, quando chamamos o setUsername(), ele altera não só o valor da variável, mas também todos os trechos que utilizam essa variável nos elementos.

Esse é basicamente o funcionamento geral do React, o que acontece muitas vezes. Na maneira antiga de trabalharmos com a web, a pessoa digitaria o nome nesse campo input e recarregaríamos a tela inteira, pegando essa variável e mandando para o back-end, renderizando tudo novamente, para então mostrar o resultado.

Com o React esse processo fica mais leve e mais fácil, sem precisar de muito trabalho manual.

**Mario:** Perfeito, Paulo. É exatamente isso que você trouxe para a galera. Quando alguém te perguntar em uma entrevista "por que o React é bom?", você pode responder isso, sobre a otimização. É isso que mostra que você conhece a ferramenta, não é só criar componentes ou usar várias libs.

Vou até deixar um vídeo aqui na descrição no qual, em menos de 30 minutos, eu refaço o React do zero. Então eu crio um arquivo Javascript e saio criando todas as coisas que fazem o React funcionar, e acho que é uma ótima referência para você que está começando a carreira e quer entender um pouco de como o React funciona por baixo dos panos. É um bom ponto de partida para você dar esses primeiros passos e ir se aprofundando, conforme for necessário e você for se sentindo mais confiante.

Acho que já conseguimos entender bem essa ideia de termos variáveis e alterarmos seus valores. Outra coisa importante é fazermos testes. O Paulo disse uma coisa muito importante: e se não usássemos o setUsername()?

Vou comentar a linha do setUsername na função PaginaInicial() e voltar a usar a variável username que tínhamos antes.

export default function PaginaInicial() {

const username = 'omariosouto';

//const [username, setUsername] = React.useState('');

return {

//...

}

Com isso, não conseguiremos mais digitar no campo que temos na página. Agora tentaremos atribuir o valor em nosso <input>.

<input

type="text"

value={username}

onChange={function {event} {

console.log('usuario digitou', event.target.value);

// Onde ta o valor?

const valor = event.target.value;

username = valor;

//Trocar o valor da variavel

//através do React e avise quem precisa saber

setUsername(valor);

}}

}

Após salvarmos, veremos que as alterações que fizermos no campo input não serão atribuídas e nada será exibido na página, ainda que nenhum erro seja exibido no navegador. Em teoria, como temos um valor constante, ele não deveria mudar.

De qualquer forma, nosso código não funciona pois é o setState() quem avisa ao React de que é necessário alterar cada um dos valores e que pedaços da página precisam ser atualizados. Por isso devemos usar o setUsername() e não só atualizar o valor diretamente.

**Paulo:** Isso pode ser estranho para quem usava Javascript puro e está aprendendo agora, já que sempre usávamos o nome de uma variável recebendo um novo valor. Aqui nós precisamos "avisar", chamar uma função, do contrário o React não ficará ciente dessa mudança de estado.

Sempre que tivermos uma variável que precisa refletir em outros pontos da aplicação, provavelmente usaremos o useState() e chamaremos pelo setter, ou não teremos o efeito desejado.

**Mario:** Perfeito, Paulo. Aqui nós conseguimos fazer essa primeira parte. Agora eu vou comentar o <input> e descomentar o TextField, que é o nosso campo de texto.

Se inspecionarmos esse elemento, veremos que por baixo dos panos ele é uma tag <input> comum. O que muda é que passaremos o value=(username) e o onChange.

<Textfield

value=(username)

onChange={function {event} {

console.log('usuario digitou', event.target.value);

// Onde ta o valor?

const valor = event.target.value;

//Trocar o valor da variavel

//através do React e avise quem precisa saber

setUsername(valor);

}}

/...

Após salvarmos, conseguiremos alterar o campo input livremente, com o React atualizando a página de acordo com essas alterações.

É legal reparar nisso. Olhar o que está sendo renderizado na tela vai te ajudar muito quando você estiver utilizando componentes criados por outras pessoas, porque às vezes você precisa ver qual o HTML que está sendo gerado para entender como determinadas coisas estão funcionando. Portanto, crie o hábito de inspecionar a página e verificar na aba "elements" o que está acontecendo, o que está renderizando e assim por diante.

No console, temos alguns erros que ocorrem porque às vezes não existe uma imagem do Github correspondente ao que foi digitado no input. Isso vai até virar um desafio no final da aula.

Paulo, estamos falando bastante do React, mas acho que faltou uma coisa, que é entrarmos no chat de fato usando o botão de "Entrar" na nossa página.

**Paulo:** Perfeito, como submeter um formulário. Como funciona isso? Nós já vimos o onChange de um componente de texto, e acho que até funciona de maneira similar com outros. Agora quero entender como o botão de "Entrar" pode redirecionar para outra tela, fazer um pop-up ou outra coisa, fazer o onChange dele, ou onSubmit que tínhamos no Javascript.

**Mario:** Perfeito. Repare que toda vez que clico em "Entrar" a página atualiza. Esse é o comportamento que você citou agora há pouco, de mandar uma requisição para o servidor e recarregar a página. Isso acontece, principalmente, se você estiver trabalhando dentro de uma tag de formulário (<form>), como podemos verificar inspecionando a página.

Se procurarmos por "form" em nosso código, encontraremos um <Box> que está como formulário (as="form"), incluindo também os seus estilos. O padrão de um formulário é sempre recarregar a página, mas quando trabalhamos com frameworks como o React e o Next, passamos a evitar ao máximo esse recarregamento total, preferindo alterar somente um pedaço da página.

**Mario:** Temos que prevenir esse comportamento padrão da página de mudar as coisas. E você comentou de submeter o formulário, o que me lembra que sempre que temos uma tag de formulário podemos usar um valor chamado onSubmit, você pode passar uma função para ele e, tal como o onChange, que é sempre que tiver uma mudança, o onSubmit vai monitorar e sempre que tiver uma submissão vai fazer alguma coisa. Vou colocar um console.log() também.

onSubmit={function () {

console.log('Alguém submeteu o form');

}}

**Mario:** Vou salvar, carregar a página e ao clicar no botão "Entrar" ele mostra rapidamente no console a mensagem do console.log().

Temos que prevenir esse carregamento. Para prevenir isso pagamos de novo as infosDoEvento. Eu falo: "infosDoEvento previne o default, por favor". Prevenindo o default ele para de recarregar a página e agora nós temos o controle de como ir para a próxima página.

onSubmit={function (infosDoEvento) {

infosDoEvento.preventDefault();

console.log('Alguém submeteu o form');

}}

**Paulo:** Porque o comportamento default do submit de um form é ir para uma URL que ele definiu como destino e se não definiu vai para a URL dela mesma.

**Mario:** Então, queremos ir para a próxima página, que seria a página de chat, <http://localhost:3000/chat>, que não existe. Se tentarmos ir para essa URL o próprio Next retorna uma página "404".

Agora, vou mostrar como criar uma nova página. No VS Code, vamos no menu explorer, à esquerda, e dentro da pasta "pages" criaremos um arquivo chamado chat.js. Dentro desse arquivo vamos escrever:

export default function PaginaDoChat() {

return (

<div?>Página do Chat </div>>

)

}

**Mario:** O Next já sabe que isso é uma página e que ele tem que renderizá-la. Podemos carregar no navegador o endereço <http://localhost:3000/chat>. Cria um arquivo e já tem uma página, não precisa se preocupar com roteamento, ele já faz isso para nós.

**Paulo:** Então, chat.js. Por ela ser a única em que fazemos export default pode ser o nome que quisermos aí onde você colocou PaginaDoChat.

**Mario:** Exatamente, eu deixei PaginaDoChat, mas pode ser qualquer nome.

É importante dizer que o Next gerencia as páginas para nós. É ele que faz as coisas acontecerem. Se quisermos mudar de página tem várias formas. Vou mostrar uma delas. Podemos inserir no onSubmit um window.location.href = '/chat';.

onSubmit={function (infosDoEvento) {

infosDoEvento.preventDefault();

console.log('Alguém submeteu o form');

window.location.href = '/chat';

}}

**Mario:** Esse é o jeito tradicional de o navegador fazer isso. Inclusive, se colarmos essa linha de código e colarmos no console do navegador, na home, ao pressionarmos "Enter" ele muda de página, vai para a página do chat.

Se eu carregar a home e clicar em "Entrar" ele vai para a página do chat também. Mas repare que ele está sempre recarregando a página durante essa transição, podemos ver que o ícone de atualizar, na área superior da página, está recarregando.

Como é o Next que gerencia essa parte de roteamento, o Next tem um recurso certo para fazer troca de páginas. No topo do arquivo index.js vou fazer import {useRouter} from 'next/router';. Vamos pegar o sistema de roteamento do Next.

Viram que tem uma semelhança em como se escreve useRouter e useState? Eles fazem parte de um padrão do React chamados "Hooks", são ganchos para você interceptar algo.

**Paulo:** E o setUsername também é um hook?

**Mario:** Sim, exatamente. Deixei para trazer agora para mostrar justamente isso. É o gancho do roteamento, o gancho de manipular o estado, o gancho de efeito. Sempre tem algum gancho no React para fazer alguma coisa. É sempre acompanhado do "use".

Então, na PaginaInicial() vou fazer const roteamento = useRouter();. Basta fazer isso para usarmos. Se dermos um console.log(roteamento), podemos ver lá no console do navegador que ele mostra uma lista informando qual é a pasta em que estamos, qual o basePath, qual a função para voltar página, um monte de coisas.

A navegação da web é como se ela fosse uma pilha, um array, estamos sempre colocando uma coisa em cima da outra. Tanto que olhando essas funções que temos para trabalhar, temos aqui o push, que serve para adicionar mais uma URL nessa pilha de navegação.

Voltando para o VS Code, vou apagar o console.log(roteamento) e dentro no nosso onSubmit vou escrever roteamento.push('/chat').

onSubmit={function (infosDoEvento) {

infosDoEvento.preventDefault();

console.log('Alguém submeteu o form');

roteamento.push('/chat');

}}

**Mario:** Fazendo isso, ao clicar no "Entrar" ele vai para a página do chat e não faz mais aquele carregamento. Podemos ver na aba "Elements" do Inspecionar o que muda nessa transição. Mudou só os pedaços que precisavam.

Aqui já fechamos o conteúdo dessa aula. Só quero trazer um adicional antes do desafio. Está vendo que aqui na página do chat a fonte do texto mudou, voltou a ficar com a serifa? O GlobalStyle não está funcionando aqui. Ele está global, mas está global na página inicial.

Precisamos colocar o GlobalStyle aqui na página do chat também. A forma correta de fazer isso, é o seguinte. O Next tem um arquivo em que ele encapsula todas as páginas que serão carregadas, ou seja, se você quer que algo realmente fique em volta de tudo, você coloca nesse arquivo. Vamos entrar no link do [Custom App na documentação do Next](https://nextjs.org/docs/advanced-features/custom-app).

O que faremos é, basicamente, copiar esse código da documentação e criar o arquivo \_app.js dentro da pasta "pages".

Esse \_app.js não vai virar uma página. Precisa fazer o export default dele e a vantagem é que tudo o que colocarmos aqui será carregado em todas as páginas.

export default function MyApp({ Component, pageProps}) {

console.log('Roda em todas as páginas');

return (

<Component {...pageProps} />

);

}

**Mario:** Essa mensagem "Roda em todas as páginas!" vai aparecer no console da página do chat e na home também. Está em volta de tudo, é um wrapper, ele carrega em volta de todas as páginas.

Importante falar que no React é muito comum, quando você instala uma lib ou quer algo genérico na sua aplicação inteira, você configurar nesse arquivo \_app.js. Por isso é importante saber que ele existe. No nosso caso, a configuração será de CSS.

Vamos recortar o trecho de código do GlobalStyle() que está no index.js e colar dentro do \_app.js, no topo do código. E no function MyApp vamos chamar o <GlobalStyle> lembrando de colocar aquela estrutura da tag vazia em volta, <> </>.

export default function MyApp({ Component, pageProps}) {

console.log('Roda em todas as páginas');

return (

<>

<GlobalStyle />

<Component {...pageProps} />

</>

);

}

**Paulo:** Todas as páginas .js vão chamar essa function que está no \_app.js. Essa função está como MyApp, mas podia ser qualquer coisa.

**Mario:** Exato. Podia ser só App, ou CustomApp, por exemplo.

**Paulo:** O importante é esse \_app.js, esse é um nome que o Next reconhece e o que estiver aí todo mundo tem acesso.

**Mario:** Lembrando que não precisamos mais chamar o <GlobalStyle> no código da PaginaInicial, podemos apagar essa linha.

Vamos ver como está nossa página inicial. Claro, tem mais detalhes para gerenciar tudo isso, mas continuaremos nas próximas aulas.

**Paulo:** Hoje mostramos os principais recursos do React e do Next. O Hook do state e do roteador são coisas que você vai mexer no React o tempo inteiro.

**Mario:** Para tudo, para abrir formulário, abrir menu etc. Tudo vai mexer com esses dois de alguma forma.

**Paulo:** Mario, por que no useRouter não foi preciso colocar "React.useRouter"?

**Mario:** Porque esse useRouter é do Next. Estamos importando direto do Next.

Então, nessa aula vimos useState, vimos que ele propaga alteração para todos os lugares; vimos o useRouter; trabalhamos com eventos, onSubmit para o formulário; onChange para campo de texto; a importância de sincronizar o value com o onChange, de usar a função setUsername para avisar o React que mudou e vimos como criar uma página nova, que basta criar um arquivo .js dentro da pasta "pages".

Eu até gosto de falar que a pasta "pages" é do Next, não é sua, tudo que está dentro dela o Next usa para montar as estruturas dele e fazer a organização da aplicação.

Paulo, qual vai ser o desafio da aula de hoje para a galera?

**Paulo:** O meu primeiro desafio - ainda não é nem desafio de implementação - é que você teste e tenha total domínio do useState, do useRouter e do que fizemos no onSubmit e no onChange, que é um pouco diferente de como faríamos no JavaScript clássico.

Quando você entender essa parte, vai começar a entender todo o mecanismo do React. Cerca de 90% do que vamos usar está muito atrelado a essa aula aqui.

Na primeira aula entendemos sobre componente, mas agora entendemos como o componente é vivo na tela, como atualiza só uma parte, como ele fica sabendo o que mudou, o que clicou e repercute isso onde precisa repercutir.

Os detalhes do CSS, de onde buscou cada coisa, se o \_app.js é do Next ou do React etc, vai ficar mais claro no decorrer da Imersão.

Agora eu quero que você se concentre nessa parte que o Mario passou hoje, porque realmente vai aparecer o tempo inteiro tanto aqui na Imersão quanto no seu dia a dia profissional.

**Mario:** E para te ajudar nessa missão, eu gostaria de desafiar você a fazer com que se o campo tiver menos de dois caracteres, você desabilita o campo e não mostra a imagem, ou seja, só mostra a imagem se tiver mais de dois caracteres no campo de texto e a validação.

**Paulo:** Essa é uma boa ideia. Acho que usa o onChange, é por aí?

**Mario:** Já está feito, só precisa criar uma variável e passar para o botão. Essa é a dica que vou dar.

**Paulo:** Perfeito. Esse já é um mecanismo muito bom para começar a validar e entender onde passa a variável, entender se vai ou não ter "use", testar e errar, que faz parte.

O que eu queria ver também, de desafio, é você customizar mais a sua tela. Ele não precisa puxar só sua imagem do GitHub, dá para puxar mais coisas. Tem uma página que dá para você acessar e pegar um JSON com vários dados. Vamos compartilhar a tela para mostrar isso?

**Mario:** Vamos. Vamos acessar a página <https://api.github.com/users/omariosouto>, você pode acessá-la inserindo o seu usuário no final. Essa página exibe um JSON com várias informações do seu usuário, nome, id, foto, a URL do perfil, os seguidores, entre outras coisas.

**Paulo:** Perfeito. Dá para brincar bastante com isso.

**Mario:** Vamos deixar materiais para consulta na descrição dessa aula. Se você quiser fazer esse desafio, pode ver um vídeo no meu canal em que mostro como pegar essas informações. Inclusive, usando o GitHub como exemplo.

**Paulo:** Estamos esperando para ver seu projeto no Discord, nos marque, use nas redes sociais as hashtags #imersaoreact, #alura, compartilhe no LinkedIn. Já tem muita gente nos marcando e estamos muito orgulhosos de ver seu trabalho e sua evolução.

Inclusive, vamos trazer algumas empresas para ver os projetos de vocês, empresas que precisam contratar não só React, mas JavaScript puro e Angular, por exemplo. Mesmo a Imersão focando no React Next, que é o maior mercado de front-end, você verá que ao mostrar sua capacidade de aprender e encarar uma Imersão da Alura do início ao fim, as empresas enxergam valor nisso.

Não é à toa a quantidade de empresas parceiras do Aluraverso e que estão conosco como clientes da nossa escola. Essas pessoas estão buscando pessoas como você, que está demonstrando sua capacidade de aprender.

Nos vemos amanhã na próxima aula. Tchau!

**Mario:** Boas-vindas à terceira aula da Imersão React da Alura! Hoje conheceremos algo que resolve um dos maiores erros de quem trabalha com front-end. Mas antes, sei que você tem um recadinho para a galera, Paulo.

**Paulo:** Eu tenho. Estou muito feliz de estarmos na terceira aula, e você precisa acompanhar a hashtag #imersaoreact e ver o que está aparecendo no LinkedIn, Twitter e Instagram. Está incrível!

Só de mostrar os primeiros passos da segunda página, as pessoas foram muito além. Parabéns para você que está elevando essa Imersão a um nível muito maior e entendendo os conteúdos que estamos apresentando. Lembrando que o useState do React e o useRouter do Next são ferramentas com as quais você trabalhará o tempo inteiro.

Se você entender bem esse funcionamento, e que não basta usar uma variável simples, esse universo do React já vai ficar mais evidente para você. Nós queremos que você realmente conheça o funcionamento do React, seus conceitos desde a base, e não só que aplique código.

Nós estamos aplicando bastante coisa, e vamos aplicar muito mais, mas queremos que você saia dessa Imersão com conhecimentos sólidos e um portfolio para mostrar para as pessoas. Nós sempre batemos nessa tecla, e é por isso que as Imersões da Alura têm esse resultado fantástico.

Inclusive estamos colocando essa ideia de portfolio dentro das formações da Alura, com sugestões de práticas, como exercitar seus conhecimentos, o que você pode criar e assim por diante, para realmente te desafiar a mostrar os seus passos. É assim que queremos que você prossiga, não é?

**Mario:** Exatamente, Paulo.

**Paulo:** Eu vou pedir aqui uma intervenção da produção para colocar mais um depoimento, para você ver onde que as pessoas que estudam conosco na Alura chegam na área de front-end. É sempre muito inspirador ouvirmos histórias reais dos nossos alunos e alunas, e tenho certeza que isso vai impactar o seu trabalho.

**Rene Sena:** A Alura me apresentou muitos conhecimentos novos. Por exemplo, essa questão de criação de rotas dinâmicas, pré-renderização de dados no servidor, assuntos que eu nunca imaginei que aprenderia ou mesmo que o front-end conseguiria fazer.

Essa é uma visão que me surpreendeu bastante, essa questão do que fazer como desenvolvedor front-end ou em outra função na área de TI, e você tem muitas possibilidades. Quando eu estava na Imersão, foi como naquele meme de explodir a cabeça das pessoas.

**Paulo:** Muito legal não é Mario? Acho que é mais um motivo para levarmos os projetos até o fim e dedicarmos essas horas por dia durante a semana da Imersão React.

**Mario:** É legal aproveitar essa oportunidade, esse momento em que você não está sozinho ou sozinha estudando. Tem várias pessoas pelo Brasil todo, e até de outros países, pessoas que falam outras línguas, assistindo e compartilhando. É uma experiência muito legal, e espero que você esteja acompanhando e criando seu portfolio. Vamos começar, Paulo?

**Paulo:** Mario, fazendo um spoiler também. Você falou de outros países, outras línguas, quem sabe a gente não traz uma imersão com uns desafios em inglês para quem está treinando ou mesmo para outros públicos? Vamos ver a recepção dessa ideia.

**Mario:** Estou com a nossa janela da aplicação aberta. Basicamente, temos o código que fizemos na aula 02, com toda a parte de estado que nós vimos. Hoje vamos reforçar esse conceito, mas primeiramente começaremos a trabalhar na página do chat, que por enquanto está um pouco incompleta...

**Paulo:** Meio feinha.

**Mario:** É, meio feinha. Voltando ao VSCode, temos a função ChatPage() no arquvo chat.js, que é a página base. Nada mudou no restante do nosso código. Agora, eu quero carregar a "base" dessa página, que seria o template inicial para começarmos a trabalhar e colocar os nossos códigos.

Vou copiar a URL da minha famosa "colinha" e abri-la no navegador. Lá, teremos um código que preparamos anteriormente com vários componentes. Ainda incluiremos outras coisas nas próximas aulas, mas essa é a parte inicial que nos ajudará a dar os primeiros passos.

Vou copiar o código disponível nessa página, lembrando que o link ficará disponível na descrição da aula, e colá-lo no arquivo chat.js.

**Paulo:** Muita coisa, Mario. Quero que você me explique o que está aí.

**Mario:** Vamos por partes. Vou salvar e abrir a aplicação o navegador para visualizarmos. Repare que fiz até uma homenagem para uma das nossas instrutoras, a Vanessa, colocando uma mensagem padrão em nosso template.

Basicamente, temos uma recepção da estrutura da página anterior. Começamos com uma caixa (Box) contendo o backgroundImage com a imagem do Matrix, com aquela chuva de caracteres; outra caixa com um background mais escuro, que é referente ao fundo do chat; além do texto "Chat" e um botão clicável de "Logout" para o usuário deslogar.

Mais adiante, temos componentes menores para facilitar a visualização. O foco daqui em diante será muito mais a parte central dessa página, onde inserimos e recebemos mensagens, entendendo como pegar essas informações.

Temos, então, um componente do cabeçalho (Header). Se segurarmos o botão "Ctrl" e clicamos sobre ele, conseguiremos navegar dentro desse arquivo, uma dica importante para quem está começando a mexer com o VSCode.

Encontramos a utilização do Header na página e, mais abaixo, a definição do componente Header(), que é "pequeno", sendo constituído por uma caixa (Box), o texto do "Chat" e o botão com um variant='tertiary', atributo que faz com que o fundo do componente clareie quando passamos o mouse sobre ele.

**Paulo:** Mario, o componente Box é aquele da biblioteca Skynex que estamos importando, certo?

**Mario:** Exatamente.

**Paulo:** Estamos estilizando esse componente com vários atributos, para ficar bonito. Novamente, se você estiver assustado ou assustada com o CSS, você pode deletar, e terá o mesmo resultado, só que um pouco mais "feio".

Já o componente Header é diferente, não é da Skynex nem de nenhuma outra biblioteca. Nós estamos criando - no caso o Mario -, como fizemos na primeira aula com o Titulo. Aqui criamos um componente Header, que seria algo como "cabeçalho", está no mesmo arquivo (chat.js) e consiste em uma função que retorna um Box.

Repare que essa função nem recebe nenhum parâmetro, é uma coisa "fixa" (algo que chamamos também de *hardcoded*), que define como serão todos os *headers*. Quem quiser, eventualmente, pode passar um parâmetro para o Header usando *props*, por exemplo.

**Mario:** Perfeito. E é legal apontar isso, pois às vezes vamos criar componentes não só para reutilizá-los, mas também para deixar mais evidente o que estamos fazendo. Assim, é possível simplificar um longo trecho de código, separando-o de uma maneira mais visível - e essa foi a ideia do nosso componente Header.

Junto com ele, temos também a parte da lista de mensagens, localizada em um componente MessageList. Parece simples, pois é composto de uma imagem, um texto, a data e a mensagem, mas analisando com mais atenção percebemos que tem bastante código de estilo, a ativação do scroll, cor de fundo e diversas tags.

Como usaremos esse código para mostrar várias mensagens, especificamente dessa parte da aplicação, acabamos deixando o componente no mesmo arquivo, removendo-o do conjunto total. Ao invés de termos cerca de 15 linhas a mais no meio do código da página, colocamos um atalho <MessageList/> que aponta para nosso componente.

Isso é bom pois também nos ajuda a navegar pelo código. Quando você abre o arquivo, consegue identificar cada parte da estrutura, e a "carga" cognitiva para analisar o que está acontecendo fica menor, facilitando o trabalho. Valeu essa tour pelo *template*, Paulo?

**Paulo:** Perfeito. Você chamou de *template*, mas está se referindo ao código do chat.js.

**Mario:** Exato.

**Paulo:** Se você reparar, nós temos somente alguns Box, um Header e um MessageList, que é o novo componente que criamos. Você pode verificar o código-fonte desse componente, que por sua vez são Box com textos.

Já os outros Box são componentes muito comuns, e é rotineiro, com o React, usarmos componentes de tercerios, como calendários, botões, designs e assim por diante. Ao final da aula, podemos dar sugestões de desafios para que as pessoas coloquem mais componentes interessantes nessa página.

A única coisa que aumentou a extensão do nosso código é o CSS, o estilo. Você pode inclusive comentar para ver o que acontece. Por exemplo, comentando o trecho de código da Box referente à mensagem do chat, ela vai perceber a sua posição e algumas outras estilizações.

A Vanessa Tonini, instrutora da Alura que já partipou de várias imersões, ficou doente durante esse período e infelizmente não conseguiu participar desta.

**Mario:** Mas deixamos uma homanegem a ela, eternizada na imersão.

Nós fizemos essa parte base do nosso chat. Eu chamo de "template" porque é assim que nomeamos na descrição da aula e no e-mail que é enviado aos alunos e alunas. Agora, Paulo, vamos pensar no que é necessário para que esse chat funcione.

Baseado nos chats que você já usou na vida, o que deveríamos ter de funcionalidades? O que o usuário fará nessa página para mandar as mensagens dele?

**Paulo:** Digitar e apertar o "enter"?

**Mario:** Boa. O usuário digita no campo textarea, aperta "enter" para enviar - até porque não temos um botão de enviar. E quando ele envia essa mensagem, ela deve aparecer na listagem junto à mensagem da Vanessa, certo?

**Paulo:** Isso, deve adicionar o texto em um array, ou em uma lista, não sei.

**Mario:** Vou anotar tudo isso como comentário em nosso arquivo ChatPage(). Eu gosto bastante de fazer isso, inclusive fazia bastante no presencial, que é deixar evidente o que o usuário vai fazer e o que nós, como desenvolvedores, precisamos para implementar isso.

Por exemplo, para o usuário digitar no campo de textarea nós precisamos criar o campo, que inclusive já está criado. Para que ele pressione o "enter" e envie a mensagem, o que precisamos fazer? Queremos salvar a mensagem e que o React perceba que ela está sendo enviada.

**Paulo:** Usaremos o onChange?

**Mario:** Boa.

**Paulo:** A gente provavelmente vai chamando o setState() para trocar a variável que é o valor da mensagem. Também precisa ter um if lá dentro para verificar se o caractere digirado é um "enter", que aí faremos algo diferente, vamos limpar a variável e jogar o valor antigo para uma lista.

Acredito que essa lista de mensagens também vai ser um state, e quando ele atualizar nós atualizaremos também a lista de mensagens colocadas. Então acho que tanto a lista quanto o campo de mensagens são states.

**Mario:** Perfeito, Paulo. Você mandou lindamente a resposta. Acho importante esse ponto, e todo mundo que está na imersão tem que conseguir ter esse raciocínio de como o usuário vai se comportar e o é necessário fazer com o código para permitir esse comportamento.

E você percebe que as nossas listas estão mais ou menos "um para um":

// Usuário

- usuário digita no campo textarea

- aperta enter para enviar

- tem que adicionar o texto na listagem

// Dev

- [] campo criado

- [] vamos usar o onchange, usa o useState (ter if para caso seja enter pra limpar a variavel)

- [] Lista de mensagens

Para que o usuário aperte "enter" e envie a mensagem, você tem que ter o raciocínio de que será necessário usar o onChange e o useState.

Eu falo muito isso, porque quando comecei a programar eu tive um trauma. Eu queria fazer um botão de *"like"*, e eu procurava no Google "como fazer um botão de *like*". É muito claro para mim que com o tempo nós mudamos a forma como pesquisamos as coisas, e conseguimos abstrar, por exemplo, que fazer um botão de *like* é adicionar um evento de clique em um botão e manipular o HTML para alterar um valor.

No React, é fazer o onClick no botão e fazer o useState. É interessante ir praticando isso, porque vai te ajudar a participar até de tarefas mais complexas. Empresas maiores têm processos de whiteboarding, por exemplo, onde, sem escrever código os devs vão discutir e tentar desenhar como será determinada funcionalidade. Isso vai trazendo mais confiança para você participar dessas discussões e ir melhorando na carreira.

Como primeiro passo, podemos começar pelo onChange e useState, que são conteúdos que abordamos na última aula e estão mais frescos na memória.

Temos o campo de texto do chat onde conseguimos digitar livremente. Vamos procurar no código esse textarea e começar a trazer as funcionalidades para ele.

Nós temos um campo de texto <TextField> do tipo textarea. Existem outros tipos, como password, number, phone, email e assim por diante - basicamente os mesmos tipos suportados pelo HTML. Temos também o CSS estilizando esse conteúdo.

<TextField

placeholder="Insira sua mensagem aqui..."

type="textarea"

stylesheet={{

width: '100%',

border: '0',

resize:'none',

borderRadius: '5px',

padding: '6px 8px',

backgroundColor: appConfig.theme.colors.neutrals[800],

marginRight: '12px',

color: appConfig.theme.colors.neutrals[200],

}}

/>

Vamos começar a trabalhar a lógica do onChange. Primeiro, teremos um valor value que será sincronizado no React, e que será a nossa mensagem.

<TextField

value=(mensagem)

placeholder="Insira sua mensagem aqui..."

type="textarea"

stylesheet={{

width: '100%',

border: '0',

resize:'none',

borderRadius: '5px',

padding: '6px 8px',

backgroundColor: appConfig.theme.colors.neutrals[800],

marginRight: '12px',

color: appConfig.theme.colors.neutrals[200],

}}

/>

Vou copiar essa mensagem e criar uma variável no início da nossa ChatPage(), lembrando de usar a sintaxe do useState: entre colchetes passaremos, primeiramente, o valor que usaremos para mostrar aquele conteúdo - no caso, mensagem -, e em seguida o método do tipo *setter* que será chamado para alterar esse valor, que chamaremos de setMensagem.

Essas variáveis receberão a chamada de React.useState(''), passando como parâmetro uma string vazia que servirá de valor inicial para esse campo.

export default function ChatPage() {

const [mensagem, setMensagem] = React.useState('');

//...

Ao carregarmos a página, recebemos um erro que não está relacionado à funcionalidade que estamos implementando. Pode acontecer na sua aplicação, pois é algo referente ao próprio Next, e existe até uma *issue* aberta sobre o problema. Isso é bastante comum em projetos *open-source*, com a comunidade colaborando para reportar e solucionar problemas.

Acontece que, no momento em que ele gera as classes automaticamente usando uma lib interna, às vezes o Next "se perde". Para que o erro deixe de aparecer, basta encerrar a aplicação e reiniciá-la no terminal.

Agora, sempre que digitarmos algo, ativando o onChange, queremos atualizar o valor de mensagem seguindo os passos que descrevemos anteriormente. Vamos adicionar esse onChange ao nosso TextField, lembrando que ele sempre recebe uma função (function).

Na verdade, é mais comum trabalharmos com um tipo de sintaxe conhecida como *arrow function*:

() => {}

Essa é uma sintaxe mais "simplificada", que pode parecer estranha à primeira vista, mas aparece bastante no dia-a-dia.

<TextField

value=(mensagem)

onChange=(() => {

setMensagem();

})

placeholder="Insira sua mensagem aqui..."

type="textarea"

stylesheet={{

width: '100%',

border: '0',

resize:'none',

borderRadius: '5px',

padding: '6px 8px',

backgroundColor: appConfig.theme.colors.neutrals[800],

marginRight: '12px',

color: appConfig.theme.colors.neutrals[200],

}}

/>

Agora eu quero dar outra dica, Paulo. Muitas vezes a pessoa não lembra como fez determinada funcionalidade. Só que se você já fez seu código, pode voltar nele para lembrar como trabalhar nesse caso. Se voltarmos nas nossas páginas até a *home* e procurarmos por "onChange", veremos que fizemos esse código e sabemos o valor, que está em const valor = event.target.value;.

Então esses comentários que fomos deixando, ajudam vocês, que estão assistindo e praticando agora, a conseguirem se achar no próprio código, em um primeiro momento, e com o passar do tempo você vai ganhando mais autonomia para procurar na documentação. Então se você já fez, tenta olhar novamente o código para recuperar a memória do dia que você fez.

Ao invés de copiar essa variável, analisaremos a ideia. Temos const valor = event.target. value e levaremos esse conceito para o código que estamos escrevendo.

onChange(event) => {

const valor = event.target.value;

Isso é uma ideia super válida, certo, Paulo? Você que já deu muitas aulas. Tentar evitar, no começo, copiar os códigos. Quando estivermos codando a funcionalidade, tentarmos sempre escrever enquanto criamos a lógica.

**Paulo:** Isso, Mauro.

**Mario:** Agora que temos esse valor, definimos ele com setMensagem(valor); e teremos definido o value, onChange e o valor. Salvamos.

**Paulo:** Podemos ver esse código de várias outras formas, certo, Mario?

**Mario:** Sim!

**Paulo:** Porque naquele trecho não precisávamos ter quebrado em duas variáveis, ao invés da *arrow function*, era possível escrever uma *function* mesmo. Tem várias formas de escrever esse trecho de código, certo?

**Mario:** Certíssimo, Paulo. Não existe uma forma de outro mágica que podemos fazer para tudo dar certo. Na verdade, dará tudo certo de várias formas, mas não tem um jeito melhor ou pior. Há diferentes formas.

Se escrevermos apenas código, não muda muita coisa. Escrevemos e a informação apenas aparece. Então podemos copiar o valor e colocá-lo em um pedaço do código. Em cima da lista de mensagens, codamos (mensagem), que é o valor do TextField, e conferir se o valor muda. Portanto, escreveremos Tá mudando o valor: (mensagem). Colocamos essa linha no meio do código, recarregamos a página e, enquanto digitamos, podemos reparar que o valor está mudando.

Essa é outra dica importante. Esse começo de aula está cheio de dicas. No React, o que acontece com frequência é a dúvida do que você poderá trabalhar: tenho uma *string* ou um número? Isso acontece até no Javascript, por não ter um intérprete *pine*, *flort* e outros, às vezes ficamos perdidos, sem saber o que usar.

Então tenta sempre colocar o valor na tela ou usar um console.log para ter clareza do que está trabalhando e, com isso, escrevemos um código melhor, evoluir a aplicação e tudo mais.

Observamos que o código está funcionando e mudando o valor. Apagaremos esse trecho do local onde colocamos e, no começo do código, Paulo, veremos que o próximo passo é usar o onChange e o useState para usar as ações. E você colocou um if para quando apertarmos “Enter”, limparmos a variável. Isso é no caso de realizarmos o *submit*, ou seja, quando apertarmos a tecla "Enter" conseguirmos enviar.

**Paulo:** Isso! Boa! Deixa de desafio ter também um botão ao invés de só o "Enter". É possível ter várias ideias.

**Mario:** Perfeito. Gosto bastante da sua ideia, mas acho que pode ser legal vermos como fazer para ter esse if. Porque caso aperte "Enter", temos que descobrir qual tecla a pessoa está apertando. Faz sentido? Temos que descobrir isso de alguma forma.

Ao conferirmos nosso código, o (event), que tem as informações do evento event: React.Change.event<HTMLInputElement> , podemos codar console.log nele e descobrir se ele diz a tecla que está sendo apertada, porque saber qual a tecla é importante. Porque estamos digitando qualquer letra, mas às vezes a pessoa aperta o "Enter". E como sabemos que a pessoa apertou o "Enter"? Então temos que saber isso pelo (event) para debugarmos juntos.

Ao digitarmos quaisquer letras, notamos que o console apresenta o *synthetic event* e no *target* temos o valor, o tipo do evento, mas não traz informação de qual tecla apertamos. Se carregarmos a página, escrevermos qualquer coisa na caixa de texto e apertarmos "Enter", percebemos que para o evento do onChange, Paulo, o código não se importa muito com qual tecla foi apertada. Ele está mais preocupado com o valor que mudou do que com a tecla que apertamos.

Se quisermos um controle mais fino da tecla que apertamos, temos que verificar outro evento do campo de texto, ou seja, o *input*. Faremos isso com o comando onKeyPress=(), que é até sugerido.

Passamos a mesma lógica com *arrow function*.

**Paulo:** Pode receber o evento e codar um console.log para vermos o que acontece.

**Mario:** Exato. Vou codar o evento.

onKeyPress=((event) => {

console.log(event);

})

Agora carregamos a página e, no console, temos o evento sintético *onKeyPress* com mais informações. Dessa vez ele mostra o código da tecla em code: "KeyA" e em key: "a" também.

Ele verifica se o "Alt" ou o "Ctrl" está apertado, então o onKeyPress é utilizado para criar atalhos de ferramentas como o Slack ou Facebook, onde apertamos "comand + K" ou "Ctrl + K" e alguma função acontece. Então essa curiosidade é legal de mostrar.

Temos, portanto, o *key* "a". Se apertamos "Enter", outro evento sintético aparece informando code: "Enter" e key: "Enter". Então fica a dica para utilizar esses valores para se basear.

Se apertarmos agora a tecla "1", teremos um *synthetic code* informando o code: "Digit1 e a key: "1". Então notamos que várias informações aparecem para nos ajudar e funções também.

Vamos voltar para o nosso código e começar a fazer o if, mostrando o console.log apenas se apertarmos "Enter", para conseguirmos limpar a variável, como o Paulo falou.

No evento sintético temos tanto o *key* quanto o *code*, mas o *key* é mais utilizado. Então podemos escrever o if(event.key) . Nas sugestões aparecerá o keyCode, que antes era muito usado, mas hoje é depreciado. Isso é legal do VS Code, Paulo. Ele está integrado com a documentação da web, chamada documentação de bolso pelo pessoal, que é a **"MDN"**.

Então todas as descrições da MDN sobre tecnologia, especificações do CSS e tudo mais, por mais que usemos React, o VS Code está integrado ao ponto de indicar as propriedades desatualizadas e quais devemos dar preferência. É bem legal usar as ferramentas atualizadas e ter esse controle.

Voltando para o código, sabemos que o *key* receberá o valor "Enter" e, se esse evento ocorrer, ele deve aparecer no console, do contrário, não aparece. Portanto, nosso *if* será

if(event.key === 'Enter'){

console.log(event);

}

Ao atualizarmos a página, enquanto estamos digitando o console não exibe nada. Ao apertarmos a tecla "Enter", aparece um evento sintético, e isso se repete todas as vezes que apertamos essa tecla. Então, aparentemente, nosso comando deu certo, mas faltou limparmos a variável.

if(event.key === 'Enter'){

console.log(event);

setMensagem(' ');

}

Recarregamos a página, apertamos "Enter" e o valor sumiu.

**Paulo:** Perdemos a variável. É só para brincar. Porque quando você apertou "Enter", a variável sumiu e ninguém guardou esse valor.

**Mario:** Exatamente. Digitamos ele e quando apertamos "Enter" ele some e ainda pula uma linha. Então agora, para codarmos do jeito certo, quando digitarmos a mensagem e apertarmos "Enter", essa mensagem precisa ser enviada para algum lugar, que é o terceiro tópico das nossas tarefas, para criarmos uma lista de mensagens. Então precisamos controlar as mensagens que temos de alguma forma.

Com o que já temos, Paulo, como você criaria essa lista?

**Paulo:** Eu criaria uma const com "listaDeMensagens" e "setMensagens". O *state* padrão dessa variável seria [], porque é uma *array* vazia, sem nenhuma mensagem. portanto, const(listaDeMensagens, setListaDeMensagens) = React.useState([]); .

**Mario:** Perfeito, Paulo. É isso mesmo.

Agora cumpriremos as duas últimas tarefas em conjunto. A segunda está mais ou menos feita, então podemos preencher os colchetes com **+/-** e a terceira será o nosso foco.

Ao invés de já integrarmos com o *message list* e fazermos vária funções, primeiro faremos aparecer só a "listaDeMensagens" na tela. Então colocaremos (listaDeMensagens) embaixo do <MessageList />, salvar e obsrvarmos o que aparece no React.

Na página não apareceu nada. Podemos até colocar um valor antes.

<MessageList />

Lista de mensagens: (listaDeMensagens);

Ao recarregarmos a página, ainda não temos nenhum retorno. Tentaremos então colocar algum valor nessa lista, para quando apertarmos "Enter", ao invés de apenas apagar, o valor ser armazenado em algum lugar.

Paulo, em qual parte do código você alteraria para adicionar esse valor a lista?

**Paulo:** Acho que será o onKeyPress, porque quando aperta o "Enter", antes de remover e fazer o setMensagem('');, guardaremos a mensagem na array "listaDeMensagem", colocando esse comando depois do console.log(event);. Contudo, tem o seguinte perigo: não farei isso, mas tenho vontade de fazer da seguinte forma.

if(event.key === 'Enter'){

console.log(event);

listaDeMensagens.push(mensagem);

setMensagem(' ');

}

Eu não tenho certeza se isso funciona ou não, porque listaDeMensagens é a variável que lemos, que teoricamente o React não saberá que alteramos ela, ou saberá, já que é um método dela? Ele me devolveu uma array mágica que, quando escrevo alguma alteração, o React sabe, ou precisa usar setListaDeMensagens(ListaDeMensagens)?

**Mario:** Vamos testar, Paulo. Digitaremos na nossa página uma mensagem qualquer e apertaremos "Enter". Repara que não aconteceu nada.

É justamente o que você disse. Essa lista é apenas para vermos, então, se quisermos adicionar algo, teremos que usar o setListadeMensagens(). E isso é algo que confunde um pouco no começo, porque como estaremos setando, teremos que passar todos os valores que tínhamos. Portanto, passaremos o *array* quanto qualquer coisa que tinha na lista e adicionar a nova mensagem que queremos trabalhar.

if(event.key === 'Enter'){

console.log(event);

setListaDeMensagens([

listaDeMensagens,

mensagem

]);

setMensagem(' ');

}

**Paulo:** Nessa parte não precisamos usar aquela sintaxe para abrir a *array* listaDeMensagem?

**Mario:** Tem! Essa sintaxe é para resgatarmos todos os itens que já estão na lista e os colocarmos na lista nova.

**Paulo:** Porque se não iríamos criar uma *array* dentro de outra *array*, que não é o que queremos. Queremos que os itens que já existem na lista e a mensagem.

**Mario:** Exatamente.

Salvamos o código, atualizamos a página, digitamos uma mensagem e apertamos "Enter". Notamos que o valor aparece em cima do campo de input. Digitei uma nova mensagem e pressionei "Enter" e vemos que tudo que digito, Paulo, ele já coloca na nossa "Lista de Mensagens".

Se inspecionarmos a página, perceberemos que há vários textinhos separados.

Podemos recuperar esses textinhos e tentar exibi-los de forma mais bonita, ao invés de apenas o conteúdo. Porque se colocarmos apenas o *array* na página, observaremos o resultado. E se quiséssemos colocar, por exemplo, em uma TAG "li" antes de preenchermos com mensagem da Vanessa?

Você tem uma sugestão o que podemos colocar aqui?

**Paulo:** Lembrei. Quando temos uma *array* e queremos mostrar todos os elementos, normalmente pensamos em forEach . O problema desse comando é que ele não retorna nada, o forEach só executa algo para cada elemento, mas não é isso que queremos. Não queremos um console.log ou um print. Se for um console.log, funciona.

<MessageList />

Lista de mensagens: (listaDeMensagens.forEach{(mensagemAtual) => {

console.log(mensagemAtual)

})}

**Mario:** Funciona. Sempre que digitarmos algo, como "oi", aparecerá no console. Temos a informação do console que na linha 60 do chat tem o "oi".

Continua sua explicação do forEach para ajudarmos quem está no curso.

**Paulo:** Agora nós não conseguimos devolver, que é nosso objetivo.

**Mario:** Exato. A ideia é que, de alguma forma passássemos por cada item e retornássemos uma informação nova no lugar daquele item.

Por padrão colocar o *array* na tela mostra o que tem nele, mas se quisermos transformar cada item, o forEach não faz isso. Se quisermos uma "li" com a mensagem atual, o forEach não faz nada.

Lista de mensagens: (listaDeMensagens.forEach{(mensagemAtual) => {

console.log(mensagemAtual)

return (

<li>

(mensagemAtual)

</li>

})}

Salvando o código acima e recarregando a página, ao adicionarmos uma mensagem no input, notaremos que nada acontece no console.

É possível confirmar isso codando no próprio console, tentando retornar a mensagem.

['oi'].forEach[(valorAtual) => {

return 'Mensagem: ' + valorAtual;

})

O forEach sempre resulta em *"undefined"*, porque o que queremos não é apenas passar por cada um, mas transformar. Queremos mapear o valor de uma lista para outra lista. Então utilizamos .map, que já nos mostra, até no console, qual será a saída.

['oi'].map[(valorAtual) => {

return 'Mensagem: ' + valorAtual;

})

**Paulo:** Coloca outra *string* nesse mara para vermos.

**Mario:** Vou digitar "oi" e "segunda string".

['oi', 'segunda string'].map[(valorAtual) => {

return 'Mensagem: ' + valorAtual;

})

Olha que genial isso, Paulo. Ajuda muito a aprendermos esse contexto quando estamos começando, porque mostra o output das duas sem nem termos apertado "Enter".

Portanto, temos o valor de entrada e mapeamos para uma nova saída padronizada, que será com "li" no nosso caso.

Então trocaremos o forEach para map no nosso código, salvamos e recarregamos a página. Ao digitarmos "oi" e apertarmos "Enter", aparecerá a mensagem no console. Digitaremos "olá", apertaremo "Enter" novamente e inspecionaremos elemento. Observaremos que as duas mensagens aparecem na nossa lista.

**Paulo:** Mario, fiquei com uma dúvida. Quando imprimimos uma *array* com console.log, ou no map, o resultado é separado por vírgula. Porque nesse caso não está aparecendo o [] com as mensagens separadas por vírgula?

**Mario:** Como assim?

**Paulo:** O "oi" e o "olá" que você mostrou no "Mensagem" dentro do chat, por que não está separado por vírgulas?

**Mario:** Perfeito, Paulo. No navegador só existe texto, então todos os valores que você coloca, com exceção do valor boleano, o React utiliza um .toString() . No caso do *array*, ele usa o toString e retira as vírgulas.

**Paulo:** É o comportamento dele.

**Mario:** É algo do React, que tira as vírgulas, deixando um ao lado do outro.

**Paulo:** Entendi, ele trabalha assim. Quando ele recebe um *array* e precisa colocar na tela, o React já faz isso.

**Mario:** Exatamente. Para o React, ter uma *array* de texto ou de componentes é a mesma coisa, a diferença é que ele procura o componente para virar TAG e o texto será um elemento de texto no navegador.

**Paulo:** Então agora o que está me incomodando é o "Enter" na *text box*, porque você aperta "Enter" e acontece uma pulada de linha horrorosa na caixa de texto. E falta acertar como faz as mensagens.

**Mario:** A quebra de linha podemos mudar dentro no nosso evento no onKeyPress, informando que sempre que quando pressionarmos "Enter", realizaremos um event.preventDefault(); , porque o comportamento padrão do "Enter" é fazer quebra de linha.

onKeyPress={(event) => {

if(event.key === 'Enter'){

event.preventDefault();

setListaDeMensagens([

listaDeMensagens,

mensagem

]);

setMensagem(' ');

}

}

**Paulo:** Entendi. Na área de texto do HTML, se aperto "Enter", ele funciona.

**Mario:** Exato. Removemos o console.log do comando, salvamos o código e recarregamos a página. Agora se digitarmos e apertamos "Enter" não temos a quebra de linha.

No console apareceu um erro que já resolveremos. Se você percebeu o erro, pode se tranquilizar porque também vimos.

Mas olha, Paulo, não temos mais a quebra de linha na caixa de texto. O legal é que, passo a passo, estamos cumprindo a primeira lista de tarefas que criamos no começo.

Já conseguimos executar o onChange, usar o if e limpar a variável do jeito certo. Nosso funcionamento já está funcionando como queríamos.

Em seguida, podemos analisar o setListaDeMensagens. Dentro desse comando, o código parece a submissão do nosso formulário. Quando enviarmos a mensagem, esse será o código que irá gerenciar essa parte.

**Paulo:** Sim. Esse a e alinha de baixo, que limpa a caixa de texto.

**Mario:** Essas lógicas, são lógicas de estado. Com elas estão acontecendo juntas, podemos separá-las e deixá-las agrupadas.

setListaDeMensagens([

...listaDeMensagens,

mensagens

]);

setMensagem('');

Talvez usaremos isso quando tivermos, por exemplo, os *stickers* ou alguma outra interação na página, essa parte da lógica do envio é interessante isolarmos em uma função e deixarmos mais próximo do topo do arquivo. Poderíamos fazer, handleNovaMensagem();. Nós chamamos dentro do onKeyPress.

onKeyPress={(event) => {

if(event.key === 'Enter') {

event.preventDefault();

handlerNovaMensagem();

}

}}

E esse handleNovaMensagem(), nós criamos mais acima, function handleNovaMensagem() {.

function handleNovaMensagem() {

setListaDeMensagem([

...ListaDeMensagens,

mensagem

]);

setMensagem('');

}

Algo que as pessoas gostam de fazer também, é: como estamos isolando uma função, passar o valor da mensagem para o handleNovaMensagem. Então, dizemos que vamos receber a nova mensagem, function handleNovaMensagem(novaMensagem) {, que nós vamos criar e também a passaremos em mensagem: novaMensagem,.

function handleNovaMensagem(novaMensagem) {

setListaDeMensagem([

...ListaDeMensagens,

mensagem: novaMensagem,

]);

setMensagem('');

}

**Paulo:** Até porque, no futuro, talvez recebamos o nome da pessoa do Chat, outros parâmetros e concentramos tudo nesta parte, pois é necessário pegar, limpar, fazer a barra de rolagem, entre outros.

**Mario:** Nós vamos fazer isso agora, é o próximo passo. Nós temos handleNovaMensagem() e o valor da mensagem que receberemos. No handleNovaMensagem(), nós passaremos o valor atual da mensagem, que é quando estamos digitando e vamos salvar.

onKeyPress={(event) => {

if(event.key === 'Enter') {

event.preventDefault();

handlerNovaMensagem(mensagem);

}

}}

Retornando ao navegador, no Chat, temos o campo "Insira sua mensagem aqui...", vamos escrever: "Ola"; "Funciona". Com o Console aberto em paralelo, inspecionando, notaremos que deu certo, funcionou.

Agora, vamos usar a sua ideia, Paulo, para atacar outro problema, que é: toda vez que renderizamos uma lista de alguma coisa no React, ele tem alguns mecanismos internos e precisa saber como identificar cada um dos itens. Como saber que cada um dos itens é único na página, para facilitar alguns processos de otimização, performance, entre outros.

Para isso, ele espera que cada item de uma lista tenha uma *prop* chamada de key, é a unique "key" prop. Se verificarmos nosso .map, poderíamos pegar mensagemAtual e passar no key o próprio valor da *string*, do texto que estamos recebendo na mensagem, passamos no key e ao verificar o Console, constataremos que ele para de mostrar o erro.

<MessageList />

{listaDeMensagens.map((mensagemAtual) => {

return (

<li key={mensagemAtual}>

{mensagemAtual}

</li>

)

})}

Se, na página, rodarmos um "oi" e um "ola", notaremos que ele parou de mostrar o erro. Porém, se digitarmos "oi" de novo, outra mensagem de erro aparece, "warning: Encountered two children with the same key", informando, portanto, que ele "encontrou dois filhos com a mesma chave". Antes de nos preocuparmos em como lidar com isso, podemos pensar na possibilidade de ter um *id*, uma lista de mensagens.

**Paulo:** Pode ser 1,2,3, isto é, mensagem 1, mensagem 2 e mensagem 3. Vale lembrar também que se trata de um *warning* e não de um erro.

**Mario:** Exatamente, é um *warning*. Ele começa a ter alguns impactos de performance se a aplicação começa a crescer muito, passa a ter muitas lógicas dentro do componente em que essa lista está, mas é importante tentarmos contornar a situação de alguma forma.

Retornando ao cógido, constataremos que a mensagem atual não pode ser apenas um texto, um valor de *string*. Cada mensagem terá: usuário da pessoa; data de envio, que é o *timestamp*; e a mensagem em si. Vamos tentar formatar para que a nossa mensagem tenha essas informações. Ao definir a nova mensagem, ela terá mais informações para além do texto.

Vamos adicionar o const mensagem =. Sempre que tivermos uma nova mensagem, ele será não só uma *string* ou uma lista de coisas, mas, sim, um item com múltiplas coisas, portanto, ele será um objeto com uma "mensagem", "*message*", ou "texto da mensagem", termo que usaremos: texto. O texto é a novaMensagem que estamos recebendo.

function handleNovaMensagem(novaMensagem) {

const mensagem = {

texto: novaMensagem,

};

setListaDeMensagem([

...ListaDeMensagens,

mensagem: novaMensagem,

]);

setMensagem('');

}

Nós também teremos a informação sobre quem está enviando a mensagem. Para isso, usaremos o "*from*", "de quem" ou "de". Por exemplo, de: 'omariosouto', ou da Vanessa Metonini, de: 'vanessametonini', que é quem está mandando a mensagem agora.

function handleNovaMensagem(novaMensagem) {

const mensagem = {

de: 'vanessametonini',

texto: novaMensagem,

};

setListaDeMensagem([

...ListaDeMensagens,

mensagem: novaMensagem,

]);

setMensagem('');

}

Agora podemos gerar um *id* nosso, informando que teremos um *id* para pegar a lista das mensagens que está mais acima no código, listaDeMensagem, e o site dela, o tamanho dela, .length. Pegando o .length, resolveríamos o problema. Podemos começar colocando zero itens, um item e continuar somando. Aliás, vamos incluir um + 1, indicando essa soma.

function handleNovaMensagem(novaMensagem) {

const mensagem = {

id: listaDeMensagens.length + 1,

de: 'vanessametonini',

texto: novaMensagem,

};

setListaDeMensagem([

...ListaDeMensagens,

novaMensagem,

]);

setMensagem('');

}

**Paulo:** Agora a mensagem não é mais uma *string*, mas, sim, um objeto que tem várias coisas.

**Mario:** Exato! Tanto que, nós apenas criamos esse trecho de código acima, não estamos usando ainda, mas se voltarmos ao Chat, escrevermos qualquer coisa, carregarmos a página e apertarmos "Enter", perceberemos que ele funciona, com texto. Se colocarmos o objeto mensagem no lugar de novaMensagem (após ...ListaDeMensagens), ele "quebra tudo", quer dizer que teremos um erro.

function handleNovaMensagem(novaMensagem) {

const mensagem = {

id: listaDeMensagens.length + 1,

de: 'vanessametonini',

texto: novaMensagem,

};

setListaDeMensagem([

...ListaDeMensagens,

mensagem,

]);

setMensagem('');

}

A mensagem de erro informa que objetos não são válidos como *React child*. Ele encontrou um objeto com as chaves "*id*", "de" e "texto". Ou seja, se temos um objeto, não podemos só renderizá-lo, precisamos pegar cada um dos valores e distribuí-los da forma correta.

**Paulo:** Ou é uma *array* de componente React, ou uma *array* de *strings*. Se não for isso, ele não aceita. Não vai transformar em texto magicamente, nem vai chamar o toString().

**Mario:** Não vai. Nós podemos pegar a mensagemAtual que está no trecho abaixo e se parece com algo que estamos recebendo.

<MessageList />

{listaDeMensagens.map((mensagemAtual) => {

return (

<li key={mensagemAtual}>

{mensagemAtual}

</li>

)

})}

Mas precisamos recordar que temos "*id*" (segundo a mensagem de erro que apareceu anteriormente), portanto, faremos mensagemAtual.id.

**Paulo:** A que está abaixo pode ser o "de", dois pontos e o texto. Concatenando, desta maneira, com o texto.

<MessageList />

{listaDeMensagens.map((mensagemAtual) => {

return (

<li key={mensagemAtual.id}>

{mensagemAtual.de}: {mensagemAtual.texto}

</li>

)

})}

O key é um mecanismo interno. Se não tivéssemos usado o key, a única coisa que aconteceria é que o React reclamaria com *warning*. O de e o texto são elementos que desejamos que apareçam no Chat, sendo que o de é a Vanessa Metonini e está *hardcoded*, e o texto é o que digitamos e apertamos "Enter".

**Mario:** Retornando ao Chat, se digitarmos "Ola" e "Sejam bem vindos a aula 3", quando apertarmos "Enter", ele funciona. Já estamos conseguindo fazer o nosso Chat funcionar de forma *offline*. Agora, Paulo, para encerrar a aula, vamos apenas migrar o trecho de código acima para dentro do MessageList.

Mas, antes, eu gostaria de comentar sobre uma questão com a qual muitas pessoas do Front-End sofrem no começo: as pessoas acham que é necessário ter o Back-End pronto para, então, escrever a funcionalidade. Na verdade, sabendo quais dados você terá e qual o design da tela, é possível criar tudo de forma estática, direto no código, e, fazendo isso, podemos só pegar depois e plugar a // Chamada de um backend para cadastrarmos algo, por exemplo.

Assim alcançamos uma flexibilidade maior para cadastrar e plugar. Na próxima aula, teremos mais clareza sobre este ponto que estou trazendo agora, mas é importante tentar praticar e entender o que usuário faz, no nosso caso, digita no campo, aperta para enviar, etc., e converter isso para a lógica necessária para integrar a tarefa que estamos lidando.

/\*

// Usuário

- Usuário digita no campo textarea

- Aperta enter para enviar

- Tem que adicionar o texto na listagem

// Dev

- [x] Campo Criado

- [x] Vamos usar onChange usa o useState (ter if para caso seja enter para limpar a variavel)

- [x] Lista de mensagens

\*/

Algum ponto a adicionar, Paulo?

**Paulo:** Não. Perfeito!

**Mario:** Agora vamos comentar o ListaDeMensagens.

<MessageList />

{/\* {listaDeMensagens.map((mensagemAtual) => {

return (

<li key={mensagemAtual.id}>

{mensagemAtual.de}: {mensagemAtual.texto}

</li>

)

})}

E fazer com que ele funcione em cima do nosso MessageList, do componente. Esse MessageList() é uma função. Tem como ele saber que a nossa lista de mensagens existe?

function MessageList() {

return (

<Box

tag="url"

styleSheet={{

overflow: 'scroll',

display: 'flex',

flexDirection: 'column-reverse',

flex: 1,

color> appConfig.theme.colors.neutrals["000"],

marginBottom: '16px',

}}

>

**Paulo:** Precisamos lembrar onde está declarada? Não, porque o listaDeMensagens está declarado dentro da function do ChatPage.

export default function ChatPage() {

const [mensagem, setMensagem] = React.useState('');

const [listaDeMensagens, setListaDeMensagens] = React.useState([]);

**Mario:** Essa é a "cereja do bolo", o toque final, da nossa aula. Se formos ao MessageList, apertarmos "Ctrl + Enter" e tentarmos dar um console.log(listaDeMensagens);, receberemos uma mensagem no Console dizendo que não existe, que não está definido, porque cada função é isolada, só tem os valores que estão dentro dela.

**Paulo:** Então, temos que passar como argumento?

**Mario:** Sim, perfeito! Teremos um props, isto é, function MessageList(props) { e um props.listaDeMensagem, ou seja, console.log(props.listaDeMensagens);. Teremos que ir em quem chama o MessageList e passar o valor listaDeMensagens={listaDeMensagens}.

<MessageList listaDeMensagens={listaDeMensagens} />

Mas, vou até fazer diferente e dizer que no MessageList o nome é mensagens.

<MessageList mensagens={listaDeMensagens} />

Fiz isso para desassociarmos o nome da variável do que o componente está recebendo. Isso é importante! Por mais que no Chat a listDeMensagens seja o nome da nossa variável, no MessageList, podemos receber qualquer nome. Não existe a obrigação de ser o mesmo nome. Tanto que se verificarmos o MessageList no Console, aparece undefined para props.listaDeMensagens. Porém, se verificarmos apenas o props, isto é, console.log(props);, no Console aparecerá "mensagens com array vazio", {mensagens: Array(0)}.

Se eu digitar algo no Chat, por exemplo, "oi", teremos no Console "mensagens com array com um item", {mensagens: Array(1)}. E assim por diante.

**Paulo:** Tenho uma pergunta. Por que se chama "mensagens" e não "lista de mensagens"?

**Mario:** Justamente porque na hora de passar o atributo, fizemos o nome à esquerda e o valor à direita. Temos que considerar sempre essa ideia de chave e valor. Aplicamos aqui a mesma lógica do objeto.

<MessageList mensagens={listaDeMensagens} />

**Paulo:** Agora pegamos o trecho de código comentado e, de alguma forma, colocamos dentro.

**Mario:** Exatamente! Temos que pegar a lista, fazer o .map e preencher os itens. Basicamente, vamos até a "ul", e, em volta da li, faremos {props.mensagens.map() =>, passamos uma função e o return dela e pegamos todo o nosso código do li.

{props.mensagens.map(() => {

return (

)

})}

O código do li é um pouco maior, então, é importante copiá-lo com cautela até chegar ao próximo </Text>, onde temos a Mensagem. Após selecionar, vamos apertar "Ctrl + X" nesse bloco de código e o colaremos dentro do return do .map. Podemos identar o código de novo usando o "Format Document", "Ctrl + Shift + P". Salvamos a página.

Retornando ao Chat e ao Console, perceberemos que a mensagem da Vanessa não aparece mais, porque o nosso *array* começa vazio. Se digitarmos "Ola", ele mostra no Console que existe um, isto é {mensagens: Array(1)}, mas também mostra o erro do key mais uma vez. Mas, nós temos nas mensagens o *id*, de quem veio essa mensagem e o texto.

de: "vanessametonini"

id: 1

texto: "Ola"

[[Prototype]]: Object

length: 1

[[Prototype]]: Array(0)

[[Prototype]]: Object

Podemos ir ao {props.mensagens.map(() => { e, das mensagens, pegar a mensagem atual, {props.mensagens.map((mensagem) => {, e incluir key={mensagem.id}.

{props.mensagens.map((mensagem) => {

return (

<Text

key={mensagem.id}

tag="li"

styleSheet={{

borderRadius: '5px',

padding: '6px',

marginBottom: '12px',

hover: {

backgroundColor: appConfig.tehem.colors.neutrals[700],

}

}}

>

Agora, se digitamos no Chat, constataremos que o erro parou. Para cada vez que digitamos, ele roda de novo o código inteiro. O motivo é que o React está sempre executando as funções para dizer que a página vai atualizar. Mais abaixo no código, vamos trocar o nome da Vanessa, {vanessametonini} por {mensagem.de}. Em seguida, o valor de Mensagem será {mensagem.texto}.

Verificando outra vez o Chat, ele está sempre jogando para cima as mensagens.

?

Tudo bem

Ola

Podemos inverter a ordem. No trecho abaixo, é possível notar que estamos colocando ...listaDeMensagens,.

setListaDeMensagens([

...listaDeMensagens,

mensagens

]);

setMensagem('');

**Paulo:** Tem que ser "mensagem vírgula lista de mensagens".

**Mario:** Exatamente! Esse é um CSS que fizemos para o *scroll*, que inverte a ordem estão aparecendo.

setListaDeMensagens([

mensagem,

...listaDeMensagens,

]);

setMensagem('');

**Paulo:** Mario, você poderia voltar lá? Foi muito rápido. Você só trocou a ordem?

**Mario:** Isso, desci o ...listaDeMensagens, e deixei o mensagem antes. Desta maneira, as mensagens no Chat aparecem na ordem correta.

Oi

Olá

Agora foi!

A causa da diferença na ordem que os elementos é o CSS que colocamos para fazer o *scroll* sempre começar de baixo, portanto, sempre estamos lendo a lista de maneira invertida. Mas, enfim, se trata de uma questão de CSS. O importante é termos a noção de que podemos colocar um item como primeiro da lista ou como último, de acordo com a ordem que fazemos o set da lista de mensagens.

Passamos com muito carinho por cada um dos pontos, como o .map. Os comentários menores permanecerão para nos acostumarmos com o código. Temos o nosso TextField mais abaixo e toda a lógica de percorrer pelos itens. Esse conteúdo e o da aula anterior é o que sempre faremos no React: pegar uma lista e renderizar na tela; fazer algum evento para customizar; cair em um estado para mudar alguma coisa.

Então, esse é o dia a dia, o "feijão com arroz", com o qual vocês trabalharão. Acho que para essa aula é só. Estudamos muitos conteúdos e agora vocês podem evoluir seus projetos.

**Paulo:** Meu desafio é colocar um botão de "Ok" do lado do "text" no Chat e, ao apertá-lo, aconteça o mesmo efeito do onKeyPress quando apertamos "Enter". Ter duas opções de fazer a mesma coisa, para que seja possível, inclusive, que os dois chamem a mesma função para adicionar mensagens. Esse é um desafio legal, nada difícil e é, como o Mario falou, grande parte do trabalho cotidiano com o React e Next.js.

**Mario:** O meu desafio é colocar um botão de apagar, "x", para conseguirmos apagar a mensagem. Esse é um desafio um pouco mais complicado. Nenhum dos desafios são obrigatórios, são apenas exercícios, maneiras de você se desafiar, mas achei que esse seria interessante, pois já temos os dados. A minha dica é que você use o método filter() do JavaScript para fazer essa deleção.

Ao invés de usar o .map para percorrer tudo, podemos usar o filter() para filtrar a lista de um jeito que ela tira o item que bate com alguma regra que você precisará descobrir qual é.

**Paulo:** Mario, acho que é muito legal, porque reforça todo o nosso aprendizado do *state*. Estudamos agora, inclusive, com *array* e notamos a existência dos mesmos problemas. Não podemos ficar fazendo "push" na *array*.

Precisamos trocar a *array* e avisar o React: sabe a *array* que eu mandava para você verificar o estado? Mudou. Agora não é mais aquela *array*, é essa. Normalmente elas são parecidas e têm um elemento a mais ou a menos.

Geralmente é isso que acontecerá, mas pode aparecer uma completamente diferente. Ter "batido" em um banco de dados do Back-End e "puxado" outra coisa completamente diferente. Por exemplo, trocar a lista dos jogadores atuais do jogo que estão online.

**Mario:** Exatamente! Eu pretendia trazer um exemplo desses, como site de lista de filmes.

**Paulo:** É importante ter bastante atenção nisso que é o mecanismo. Repare que não ficamos mexendo no CSS do *Box*, pois o que importa estudar e reforçar é o funcionamento do React, *uses*, etc. Nós não nos concentramos no Back-End ou esperamos que ele devolvesse, porque é possível fazer o funcionamento na hora em que o Back-End ficar pronto.

Na maioria dos casos, funciona desenvolver o Front-End inteiro ou quase inteiro sem o Back-End. Basta deixar *arrays* preparados que, em breve, serão "puxadas" de um HTTP, pegamos o JSON e devolvemos a *array*. Nem ficaremos sabendo se o Back-End é em Java, pouco importa.

**Mario:** Perfeito, Paulo! Espero que tenhamos conseguido passar essa ideia para você que está assistindo e, junto a isso, gostaria de dizer que é importante entender as estruturas básicas do JavaScript, ter clareza de: quando temos uma *string*, um objeto; qual é a chave e qual é o valor do objeto; que objeto pode ter outro objeto, uma *string* ou um booleano; o que é um *array*.

Enfim, ter clareza desses valores primários é a base. Nas primeiras aulas de JavaScript estudamos esses valores e conhecê-los bem fará total diferença no trabalho com o React, porque é possível "debugar", usar o console.log(), "printar" na tela.

Recomendo que você use esses recursos visuais a seu favor, porque isso é o mais legal do Front-End, poder ver o que você está fazendo e ter esse *feedback* rápido, diferente do Back-End, em que seria necessário fazer um ambiente de testes ou algo mais complexo a depender do caso.

**Paulo:** Te convido a marcar a Imersão React no seu projeto para aparecer na vitrine que linkamos nos e-mails, enfim, para que você apareça na galeria de projetos incríveis.

Também te convido a conversar conosco no LinkedIn, no Instagram e mostrar para a comunidade do Brasil de React e Next.js o que você está fazendo e o que nós, enquanto Alura, estamos fazendo, porque certamente isso ajudará outras pessoas a se animarem com a carreira, com o Front-End e com o projeto. Isso é fundamental!

**Mario:** Então, não deixe de nos marcar! Até mais!